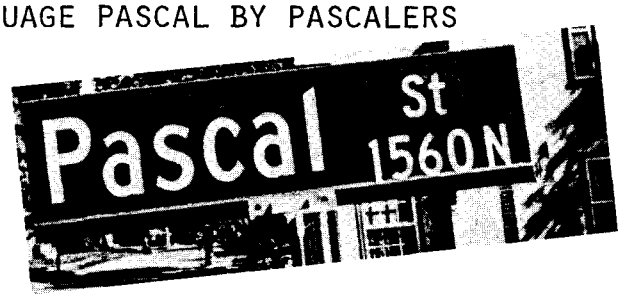
PASCAL USER'S GROUP

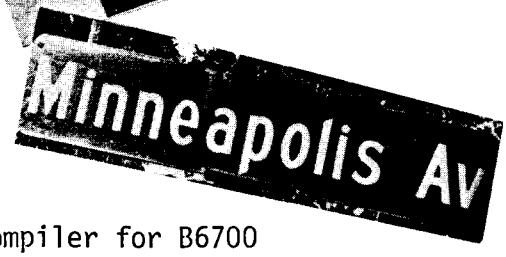
Pascal News

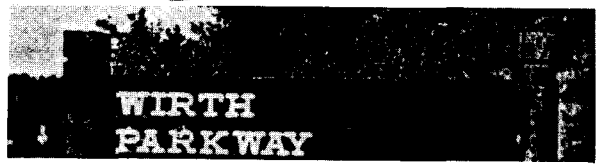
October, 1979 Number 16

SPECIAL ISSUE ON THE PASCAL VALIDATION SUITE

COMMUNICATIONS ABOUT THE PROGRAMMING LANGUAGE PASCAL BY PASCALERS







EX LIBRIS: David T. Craig
736 Edgewater
[#_____] Wichita, Kansas 67230 (USA)

POLICY: PASCAL NEWS (79/09/01)

* Pascal News is the official but informal publication of the User's Group.

  Pascal News contains all we (the editors) know about Pascal; we use it as the vehicle to answer all inquiries because our physical energy and resources for answering individual requests are finite. As PUG grows, we unfortunately succumb to the reality of (1) having to insist that people who need to know "about Pascal" join PUG and read Pascal News - that is why we spend time to produce it! and (2) refusing to return phone calls or answer letters full of questions - we will pass the questions on to the readership of Pascal News. Please understand what the collective effect of individual inquiries has at the "concentrators" (our phones and mailboxes). We are trying honestly to say: "we cannot promise more than we can do."

* An attempt is made to produce Pascal News 3 or 4 times during an academic year from July 1 to June 30; usually September, November, February, and May.

* ALL THE NEWS THAT FITS, WE PRINT. Please send material (brevity is a virtue) for Pascal News single-spaced and camera-ready (use dark ribbon and 18.5 cm lines!).

* Remember: ALL LETTERS TO US WILL BE PRINTED UNLESS THEY CONTAIN A REQUEST TO THE CONTRARY.

* Pascal News is divided into flexible sections:

  POLICY - tries to explain the way we do things (ALL-PURPOSE COUPON, etc.).

  EDITOR'S CONTRIBUTION - passes along the opinion and point of view of the editor together with changes in the mechanics of PUG operation, etc.

  HERE AND THERE WITH PASCAL - presents news from people, conference announcements and reports, new books and articles (including reviews), notices of Pascal in the news, history, membership rosters, etc.

  APPLICATIONS - presents and documents source programs written in Pascal for various algorithms, and software tools for a Pascal environment; news of significant applications programs. Also critiques regarding program/algorithm certification, performance, standards conformance, style, output convenience, and general design.

  ARTICLES - contains formal, submitted contributions (such as Pascal philosophy, use of Pascal as a teaching tool, use of Pascal at different computer installations, how to promote Pascal, etc.)

  OPEN FORUM FOR MEMBERS - contains short, informal correspondence among members which is of interest to the readership of Pascal News.

  IMPLEMENTATION NOTES - reports news of Pascal implementations: contacts for maintainers, implementors, distributors, and documentors of various implementations as well as where to send bug reports. Qualitative and quantitative descriptions and comparisons of various implementations are publicized. Sections contain information about Portable Pascals, Pascal Variants, Feature-Implementation Notes, and Machine-Dependent Implementations.

* Volunteer editors are (addresses in the respective sections of Pascal News):

  Andy Mickel - Outgoing editor; Rick Shaw - Incoming editor
  John Eisenberg - Here and There editor
  Rich Stevens - Books and Articles editor
  Bob Dietrich and Gregg Marshall - Implementation Notes editors
  Jim Miner and Tony Addyman - Standards editors
  Andy Mickel and Rich Cichelli - Applications editors
  Jenny Sinclair and Rick Marcus - Tasks editors

- - - - - - ALL-PURPOSE COUPON - - - - - - (01-Dec-79)

Pascal User's Group, c/o Rick Shaw
Digital Equipment Corporation
5775 Peachtree Dunwoody Road
Atlanta, Georgia 30342 USA

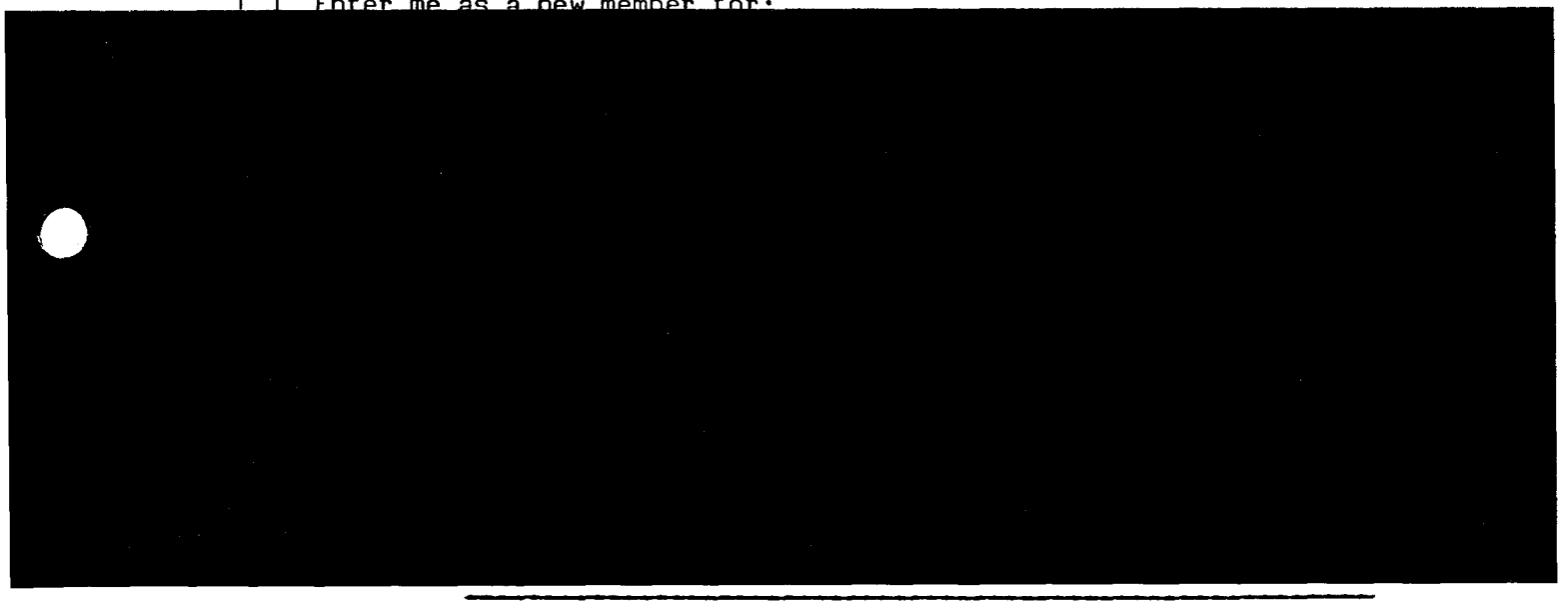
NOTE

- Membership is for an academic year (ending June 30th).
- Membership fee and All Purpose Coupon is sent to your Regional Representative.
- SEE THE POLICY SECTION ON THE REVERSE SIDE FOR PRICES AND ALTERNATE ADDRESS if you are located in the European or Australasian Regions.
- Membership and Renewal are the same price.
- The U. S. Postal Service does not forward Pascal News.

- -

[ ] 1 year ending June 30, 1980

[ ] Enter me as a new member for:



_____

ENCLOSED PLEASE FIND:	$	
	A$	
	£	____.____

NAME _____

ADDRESS _____

_____

_____

PHONE _____

COMPUTER _____

DATE _____

JOINING PASCAL USER'S GROUP?

- Membership is open to anyone: Particularly the Pascal user, teacher, maintainer, implementor, distributor, or just plain fan.
- Please enclose the proper prepayment (check payable to "Pascal User's Group"); we will not bill you.
- Please do not send us purchase orders; we cannot endure the paper work!
- When you join PUG any time within an academic year: July 1 to June 30, you will receive all issues of Pascal News for that year.
- We produce Pascal News as a means toward the end of promoting Pascal and communicating news of events surrounding Pascal to persons interested in Pascal. We are simply interested in the news ourselves and prefer to share it through Pascal News. We desire to minimize paperwork, because we have other work to do.

- American Region (North and South America): Send $6.00 per year to the address on the reverse side. International telephone: 1-404-252-2600.
- European Region (Europe, North Africa, Western and Central Asia): Join through PUG (UK). Send £4.00 per year to: Pascal Users' Group, c/o Computer Studies Group, Mathematics Department, The University, Southampton SO9 5NH, United Kingdom. International telephone: 44-703-559122 x700.
- Australasian Region (Australia, East Asia - incl. Japan): Join through PUG(AUS). Send $A8.00 per year to: Pascal Users' Group, c/o Arthur Sale, Department of Information Science, University of Tasmania, Box 252C GPO, Hobart, Tasmania 7001, Australia. International telephone: 61-02-23 0561.

PUG(USA) produces Pascal News and keeps all mailing addresses on a common list. Regional representatives collect memberships from their regions as a service, and they reprint and distribute Pascal News using a proof copy and mailing labels sent from PUG(USA). Persons in the Australasian and European Regions must join through their regional representatives. People in other places can join through PUG(USA).

RENEWING?

- Please renew early (before August) and please write us a line or two to tell us what you are doing with Pascal, and tell us what you think of PUG and Pascal News. Renewing for more than one year saves us time.

ORDERING BACK ISSUES OR EXTRA ISSUES?

- Our unusual policy of automatically sending all issues of Pascal News to anyone who joins within a academic year (July 1 to June 30) means that we eliminate many requests for backissues ahead of time, and we don't have to reprint important information in every issue--especially about Pascal implementations!
- Issues 1 .. 8 (January, 1974 - May 1977) are out of print.
  (A few copies of issue 8 remain at PUG(UK) available for £2 each.)
- Issues 9 .. 12 (September, 1977 - June, 1978) are available from PUG(USA) all for $10.00 and from PUG(AUS) all for $A10.
- Issues 13 .. 16 are available from PUG(UK) all for £6; from PUG(AUS) all for $A10; and from PUG(USA) all for $10.00.
- Extra single copies of new issues (current academic year) are: $3.00 each - PUG(USA); £2 each - PUG(UK); and $A3 each - PUG(AUS).

SENDING MATERIAL FOR PUBLICATION?

- Your experiences with Pascal (teaching and otherwise), ideas, letters, opinions, notices, news, articles, conference announcements, reports, implementation information, applications, etc. are welcome. "All The News That's Fit, We Print." Please send material single-spaced and in camera-ready (use a dark ribbon and lines 18.5 cm wide) form.
- Remember: All letters to us will be printed unless they contain a request to the contrary.

UNIVERSITY OF MINNESOTA
TWIN CITIES
University Computer Center
227 Experimental Engineering Building
Minneapolis, Minnesota 55455

Editor's Contribution

Special Issue

This special issue on the Pascal Validation Suite was prepared primarily by Jenny Mizielinski (now Jenny Sinclair) and Arthur Sale of the Department of Information Science at the University of Tasmania. We owe special thanks to Jenny because she does most of the work for PUG Australasia, and she should have been listed as a "tasks editor" long before now--in fact as far back as 1977!

The Validation Suite represents a valuable weapon in Pascal's arsenal because it provides a common measuring instrument for standards conformance. As has been said before, Pascal now joins a small and elite group of programming languages which has such a collection of test programs.

Note that the Validation Suite is copyright, but that it can be obtained very easily and inexpensively from three (3!) worldwide distributors! ANY user of ANY Pascal compiler ANYWHERE should not pass up this opportunity!

Rick Shaw

Next issue (PN#17) will be Rick Shaw's first issue as editor and is now scheduled for December. (This issue is my last one as editor.) Please don't be alarmed that Rick works for DEC. He will keep PUG and DEC strictly separated. (Besides he is just as funny and crazy a person as I am!) As I said in my editorial in PN #15, Rick is a capable administrator (whereas I am not good at delegating responsibility), and he has the luck of being in a nice work environment at DEC's Atlanta Regional Office with ready access to clerical facilities, etc. We were able to recruit section editors (listed on the inside front cover)to whom Rick can now distribute the work of Pascal News. Decentralization is manditory if Rick is to survive my fate. Good luck, Rick!

About a year and a half ago it would have been hard for me to say goodbye to Pascal News. Now it is really easy! I'm really weary and "burned-out" having worked hard all year--even after having said in issue #13 that I was tired of doing the job. Also I'm comforted that as of about a year ago,it became undeniably obvious that together all of us Pascalers permanently established Pascal as a major programming language--not just "another" language. All progress since then has been and will be pleasant dividends.

Rick's volunteering to be editor for 2 years comes just in the nick of time: a reluctant editor such as myself doesn't contribute to the quality of Pascal News. Rick will provide fresh ideas whereas I'm running out of ideas.

I feel relieved to be rid of the day-to-day responsibility for Pascal News and PUG. (However I intend to help Rick every way I can.) I do admit that working on all phases of PUG and Pascal News has made me a better person. I had the privilege to experience the processes of organizing, accounting, budgeting, editing, filing, printing, archiving, implementing ideas, pasting-up, publishing, planning, mailing, banking, maintaining mailing lists, juggling details, coordinating events, reading faster and writing better, and talking and working and negotiating and learning with other people!

Thanks

It's the honest truth: we've received hundreds of encouraging and favorable comments about Pascal News. It was truly gratifying to receive nice words this year when our hopes were dim and spirits were down. But then it is only appropriate to thank everyone who contributed material and ideas to Pascal News (by sending them in) and made the whole effort possible. Regular contributors were especially valuable. (As an example there is no reader of Pascal News who doesn't know Arthur Sale. He is a Pascal "folk-hero" because his prolific efforts are accompanied with an unforgettable signature and the end-of-the-earth Tasmanian letterhead.) I've been much less an editor than a collector and organizer of information, and I would like to say "Thanks!" and encourage all of you to keep sending in information no matter how small. Unfortunately, I'm sure we still only know less than half of the news concerning the use of Pascal!

We (especially myself) are indebted to the people whose names are listed below. They volunteered their time, energy, and enthusiasm over the last 4 years directly producing and distributing Pascal News (listed chronologically):

John Strait 1976	Herb Rubenstein 1977
Christi Mickel 1976, 1977, 1978	Arthur Sale 1977, 1978, 1979 (Aus.)
Tim Bonham 1976, 1977, 1978	Jenny Sinclair 1977, 1978, 1979 (Aus.)
Judy Mullins Bishop 1976, 1977 (U.K.)	Rich Cichelli 1978, 1979
Tony Gerber 1976, 1977 (Aus.)	Scott Bertilson 1978, 1979
Carroll Morgan 1976, 1977 (Aus.)	Steve Reisman 1978
Sara Graffunder 1977, 1978	Liz Karl 1978
Jim Miner 1977, 1978, 1979	Jerry Stearns 1978, 1979
John Easton 1977, 1978	Kay Holleman 1978
David Barron 1977, 1978, 1979 (U.K.)	Rick Shaw 1978, 1979
Rick Stevens 1977, 1978, 1979	Tim Hoffmann 1978
Tony Addyman 1977, 1978, 1979 (U.K.)	Rick Marcus 1979

How do we put together an issue of Pascal News?

Invariably the process begins by catching up (1) on the mail. This means opening an accumulation of what used to be 2-4 weeks worth in the early days of PUG to 2-22 weeks worth recently (I'm talking about trays of mail 1 or 2 meters long!). The mail must be separated (2) into new subscriptions, renewals, inquiries for information, changes of address, incorrect payments (returned), purchase orders without prepayment (returned), miscellaneous queries, and material for publication in Pascal News. (To keep our files uniform and organized we manually fill out an All-Purpose Coupon for new subscriptions and renewals; for requests for old backissues, we manually write out an address label.)

The money must be deposited (3) and accounted for (4). New members and renewers must be keyed into the data base (5) and then checked for errors (6). Back issues are mailed (7). The roster increment is run off (8), and the All-Purpose Coupons with tidbit comments are photocopied (9) for the Here and There editor.

The material for Pascal News is gathered together in a pile (10) and then sorted (11) into regular categories (Here & There, Open Forum, etc.). The Implementation Notes section is preprocessed (12) (outlined) and given to the Implementation Notes editors. The Books and Articles section is treated in the same way.

The Articles section is planned and received-dates added (13). The Open Forum section is planned (14). At this point all parts of the issue are attacked (15) at the same time including the subsections of Here & There not mentioned and the Applications section. The cloud which hangs over the process is writing the editorial (16) which delays actual page layout and pasteup of the rest of the issue.

When the camera-ready copy of the editorial and everything else is ready (or nearly so), paste-up with rubber cement on large computer-listing paper begins (18). Each sheet of large paper was previously titled and page-numbered (17) in a typewriter. The completed original is photocopied (19) to produce the 2 copies for PUG(UK) and PUG(AUS) to print from. It is then sent (20) to the printer together with a print order.

Unfortunately, these events don't always occur in this order, thus creating synchronization problems. Needless to say we are always alert for news about Pascal in other journals and from people who call on the phone.

Editor's Contribution

In Closing

As an escape clause, I've always listed: "as well as the ideas behind Pascal" together with "promoting the use of Pascal" as a purpose of Pascal User's Group. We all know that Pascal is not a perfect language, but that it best embodies the ideas of the structured-programming revolution of the 1970's.

Acceptance in the United States has been the icing on the cake and was crucial to Pascal's success as a popular programming language (sorry, ALGOL-68!). So if you stop to think, it is important to note that the Pascal movement in the United States was spearheaded by George Richmond (with some initial help by Lyle Smith, a friend of Niklaus Wirth) at the University of Colorado Computer Center primarily during the years 1972 to 1975.

The effort has been continued by Pascal User's Group via Pascal News by communicating "vast quantities of information" from late 1975 to present. Pascal News and Pascal User's Group (that is, all of us!) succeeded in centralizing authority for Pascal's acceptance, development, and standardization.

What we have done through the medium of Pascal News which was not being done (and probably could not have been done) by any other journal was to openly advocate the superiority of the principles behind Pascal. Perhaps we succeeded in shaking up enough people to accelerate rationality in programming and sensibility in computing by a number of years.

We oversaw a political process and interjected some self-fulfilling propheses to keep the action rolling. Inevitably we all were affected by the spectacular outcome ourselves!

Andy

- 1979/10/21.

DON'T FORGET TO RENEW YOUR PASCAL NEWS SUBSCRIPTION.

digital

October 23, 1979

PUG is not dead!

The first question you all probably asked yourself when you heard the news is: "How will it change?" Well, the answer is: "Not much at all!". It is going to be the same old PUG you grew to love and respect. With the same editorial policy, and the same informal approach to publication. (But it will come out four times a year, that's my only promise!) The only noticeable change will be the Editor and who does all the work. I am not a human dynamo like Andy, so I have had to enlist the aid of many volunteers (see Andy's column) to help me with all the work that used to be done almost single handed. If the next few issues are not as slick as you are used to from PASCAL NEWS, please bear with us. It may take one or two issues to get it right. Number 17 will be out by the end of the year and we hope to have Number 18 published by the end of February.

As a closing note, I would like to ask that all of you start using the new All Purpose Coupon published in this and subsequent issues of the News. It will speed the transition up tremendously between Andy and myself.

See you all next issue. Long live PUG!

Rick

Rick Shaw

RS/cgf

DIGITAL EQUIPMENT CORPORATION, 5775 PEACHTREE-DUNWOODY ROAD, SUITE 100D, ATLANTA, GEORGIA 30342
(404) 252-2600

Introduction to Special Issue

The Pascal Validation Suite - Aims and Methods

by A.H.J. Sale,
Department of Information Science,
University of Tasmania.

1979 July 13

Once upon a whisper-time - so it is said but who would believe it? - long before the Minnipins reached the Land Between the Mountains, the Glocken of Then played upon his bells and the beetle-bores, which had come to infest the countryside, fell upon their backs and waggled their legs for a space and died. But the smell of dead beetle-bores was great, and the bells could do nothing about that, and this I believe.

Pretend-story, told by Glocken to Glocken to Glocken, from Then to Now.
(by Carol Kendall - The Whisper of Glocken)

1. Definition

A Pascal validation suite? What is that? Ignoring the facetious definitions, such as a suite of motel rooms where a Pascal salesman wines and dines clients and promotes the features of the language, there are still a lot of possibilities.

It might be a set of programs that check whether some other (input) text is a valid Pascal program or not. (This may run into the halting problem, well-known to be incomputable). But it isn't. The shortest definition I can supply which describes the validation suite I am talking about is:

> *The validation suite provides programs and procedures whereby the correctness (or otherwise) of a Pascal processor may be tested.*

2. Syntax + Semantics

There are two key words in this definition which have been carefully chosen, and which deserve consideration. Firstly, the definition encompasses a Pascal *processor*, not a compiler. The definition therefore covers processors that compile native machine-code and run it, processors that utilize an intermediate code and an interpreter (for example the P-compiler), direct execution systems and pure interpreters.

But more importantly, the term processor encompasses both the analysis of the source text of a program on an execution system. I am not interested solely in determining that a compiler is "correct", whatever that means, but in determining that the *compiler+machine* pair is correct. To take a very simple example, the following program fails on some processors:

```
b:=true;  c:=false;
if (b=(not c)) then writeln('PASS')
               else writeln('FAIL');
```

What goes wrong? On analysis of the failures, the compiler seems to generate good, correct, code. Giving P-like code as an example, the test compiles to:

```
LOAD  b      {to stack}
LOAD  c      {to stack}
NOT          {logical inversion of c}
EQUAL        {test equality, leave logical result}
BRFL  ...    {branch if false}
```

The flow usually resolves into a failure of the compiler assumptions. The NOT instruction perhaps does not do the transform *true* ↔ *false*, but does a whole-word bit inversion. Coupled with the action of an equality test, of course it may then fail! The bit pattern resulting from inverting the false-pattern is not necessarily the same as the true-pattern.

Resolution of this problem can take several paths, presuming that the machine architecture is fixed. (This assumption is false for interpreters, of course). The simplest resolution is to ensure that every occurrence of the not-operator results in code that allows only the bit-patterns 00....00 and 00....01 to be generated, as is the usual representation of boolean values in more than one bit. In the Burroughs B6700 this can be achieved by emitting the instructions:

```
LNOT         {wordwise logical inversion}
ISOL(0,1)    {isolate the 0-th bit}
```

Another resolution hinges on the manner in which the branch tests are done. Suppose that the machine, when faced with a complex logical expression, generates a sequence of branch instructions which lead eventually to loading a true or false value (or a branch if that is required.)

A third resolution permits other representations for true and false (perhaps including don't care bit positions), but realizes that an unusual representation may require special coding

(a) at conditional branch points
(b) wherever ordering is important (eg. for parameters of ord, succ and pred applied to boolean operands), and
(c) in 'relational expressions' involving booleans, which may be turned into something other than number comparisons (eg. exclusive-or).

For example, in the Burroughs B6700 the representations ??..?0 and ??..?1 would suffice as the conditional branch instructions sense only the right-end bit, and consequently complex code would only be needed for the relatively unusual case where the ordering of the values was relevant. But this is not a treatise on implementing booleans. Suffice it to note that testing a processor involves the semantics as well as the syntax, and the machine as well as the compiler.

3. Ultimate futility or useful weapon

The other carefully chosen word in the definition is *correctness*. Many people have pointed out that testing cannot prove a program to be correct; it can only uncover bugs in it. Since a compiler is a program, and only part of a Pascal processor, is testing therefore an exercise in futility that we should abandon in favour of proving the compiler-machine pair to be a correct implementation of Pascal?

Of course, yes, if we were capable of it. But proof procedures are still human processes, and still subject to error and oversight. Therefore, even proved systems should be subject to testing in order to uncover weaknesses of the proof or oversights. It will be obvious that this is necessary when it is realized that Pascal-P, despite its heritage of careful design, its origin in Zurich, and its extensive testing, still has errors in it which can be detected in most of its descendants.

The validation suite is a set of programs, methodically assembled (and otherwise) which therefore exercise a Pascal processor fairly thoroughly and hopefully uncover many of the flaws in its design or mistakes and deviations from the intended actions.

So much for the theory. How does it work in practice?

4. Conformity

The most obvious set of tests to incorporate are those set down in the Pascal Standard as required of a Pascal processor, or implied to be. These were generated by systematically working through the Standard and wherever it said something was allowed, writing a program to check that it was. All such programs are, of course, standard Pascal.

Since the program *should* execute, it is arranged to print 'PASS' if it works correctly. Some such tests check semantic details, and incorporate run-time checks that lead to failure messages. By the nature of the test, several conformity checks can be included in one test program. Any failure is sufficient to show up a flaw: successful programs exercise all tested features.

However there are three problems: iteration, depth and irregularity.

The first arises with iterative/recursive syntax, or axiomatic recursive semantics. Clearly no program can test all cases of a potentially infinite system! This is tackled by the "one, two, many" principle, named after the primitive enumeration systems of nomad Bushmen in southern Africa. The test will include a minimum case of the construct, probably its successor, and a case which is a small multiple of iterations/recursions, and quite plausible. For example, for identifiers in a *var* declaration:

```
program txxx(output);
var
    onlyone : integer;
    first,second:boolean;
    x0,x1,x2,x3,x4 : char;
begin
    ....
```

Since no processor in a finite computer can provide for infinite recursion/iteration, these tests establish a *prima facie* case that the construct is present and works for small instances up to some unknown limit. To establish that the limit is sufficiently high that it will almost never prove to be a problem - what I have called a *virtual infinity* - another program is written which has a repetition slightly (and only slightly) beyond a plausible maximum. Perhaps 100 cases might be enough for some things, and 20 for other things. Such tests are regarded as not being compulsory, and form part of the *quality* measurement category. (Nevertheless, a processor may fail even a quality test by getting itself knotted, or giving an erroneous diagnostic, or behaving in an unexpected fashion...)

The third problem is irregularity. Successful execution of a test is no guarantee that the same feature will be handled correctly in a different context. Unfortunately there is no way to know all possible context changes that might affect the outcome, and the designers of the test programs have had to draw on their knowledge of machines and implementation techniques to explore this difficult area. A good example is the implementation of type boolean. Clearly, from our earlier discussion, relational operators must be tested separately for booleans and other scalar types.

Another example which maybe should be in the test package, but isn't yet, is the following

```
    csquared:=2.0*alpha*beta,
    if csquared = a*a+b*b then
        {writeln(a,b, sqrt(csquared))};
    ...
```

A particular processor noted that *csquared* was used immediately after its assignment and modified the code to leave this value on the run-time stack to eliminate the fetch. A separate optimizing routine realized that the if-statement had no effect once the debug print was factored out, and deleted the whole of the if-statement code. Result: the stack grew every time this code was executed.... Such interactions require cunning and experience to deduce, let alone devise tests for their presence.

5. Deviance

Besides saying what ought to be allowed, the Pascal Standard says what ought not to be allowed, both explicitly and implicitly. If a processor allows such constructs, it is either a deviant processor and ought to be fixed, or it embodies some deliberate extension which ought to be documented.

Note that it is pointless trying to detect all possible extensions: they form an infinite set! In any case this is not the purpose of deviance tests. Their purpose is to detect the many traps and failures to enforce reasonable restrictions which compiler implementors can unwittingly place in the way of users.

Some deviance tests are directly suggested by the Pascal Standard. Good examples are afforded by the restrictions placed on for-statement control-variables. But by far the larger group are not. These rely on the test program writers' intelligence and knowledge of compilers and on the ability or experience to recognise possible problem areas. Two examples will illustrate this.

The first was derived from some experience with an optimizing compiler which generated different code for

```
j div k
```

depending on whether *k* was a constant power of two or not, or so the documentation said. This immediately challenged the hardware designer in me to check this out, and challenged my software designing side to prove it correct. Since the optimization relied on a property of the integer representation, it was potentially possible for it to fail. Sure enough, on this compiler, it did: *j div k* returned different results for the same value of *k*, solely depending on whether it was a constant or not.

The second case grew out of a knowledge of the workings of the compatibility algorithm of most Pascal compilers, and some strict wording in the Standard. Strictly

```
type
    digitstring=packed array[1..10] of '0'..'9';
```

is not a 'string' type, and not compatible with the usual quoted string constant. Thus

```
var
    d : digitstring;
begin
    d:='0123456789';
```

is not correct Pascal: it is a deviation or an extension. Of course, compilers that allow it as an extension ought to do it properly (ie. consistently with the axioms of Pascal), so this naturally gives rise to the attempt:

```
d:='ABCDEFGHIJ';
```

which is plain impossible.

Enough of examples. It will be obvious that every deviance test program is non-standard, and that they should mostly fail to execute to completion. Unlike conformance tests this means that each deviance test tries one and only one deviation. Otherwise one failure might mask another, more serious. Conformance and deviance tests make up the majority of the tests (71% of the current suite), and do the bulk of the exercising of the compiler.

6. Errors and implementation features

The Pascal Standard also specifies a number of situations which are specified to be errors, implementation-dependent, or implementation-defined. Each of these gives rise to a test or tests that evoke the feature so specified.

In the case of errors and implementation-dependency, the purpose is to enable documentation of the error-handling capability of the processor. Hopefully all errors are detected; in practice some of them survive into production software and may even become part of programmer's assumption kit... All such programs are directly suggested by the Standard, evoke one and only one error, and are standard Pascal except for the error.

Naturally, implementation-defined features should be documented, and the corresponding tests attempt to do this by a variety of techniques. Some, like printing the value of maxint, are simple standard Pascal. Others, like trying to detect the significance limit of identifiers, rely on assumptions about the nature of the lexical analysis and scope and are clearly not standard. The inclusion of these tests is really to improve the documentation of Pascal processors.

The number of tests in those categories is small and unlikely to change much. The specific things they test are more or less fixed by the Standard.

7. Quality

All the preceding tests address the problem of whether the processor conforms to the Pascal Standard and, where necessary, how. It is also relevant to ask whether the implementation measures up to some quality standards, and the validation suite therefore contains tests which attempt to assess *quality*, however you define it.

As you might expect, these tests form a motley group. Some attempt to assess whether the processor provides some hard limits that are likely to annoy in a particular direction, or whether the limits are at virtual infinity. Others attempt to assess accuracy of mathematical functions such as sqrt(c). What they all have in common is a requirement for greater analysis of the results and a value judgement.

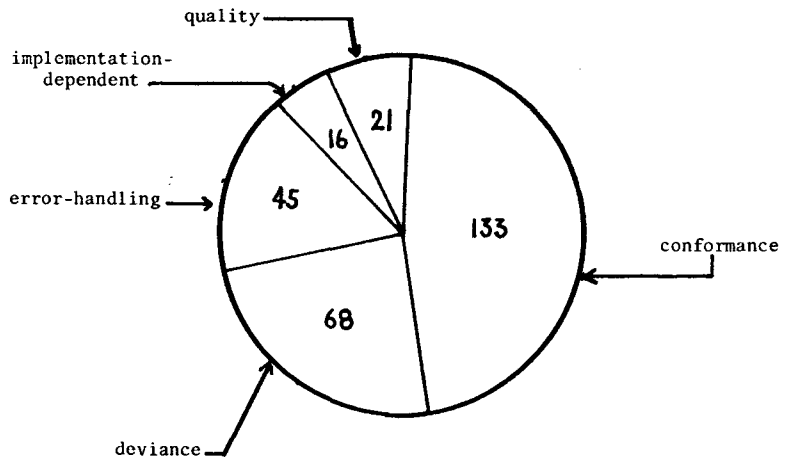
Of course, a processor may fail even a quality test by totally mishandling it. For example, if the compiler goes into an infinite loop, or gets its lexical levels screwed up. But this is not the intention, and is rare.

This group of tests is likely to have the greatest growth rate in future revisions of the validation suite. For obvious reasons, the first release version has concentrated on correctness, and while there may be some growth in tests for conformity and deviance as we understand the problem better, it is already apparent that the suite lacks quality tests in several areas:

(a) tests that can be timed,
(b) tests whose space utilization is measurable,
(c) tests of compiler diagnostic ability,

and there are no doubt more.

The breakdown of programs in version 2.1 by class is shown below.



BREAKDOWN OF PROGRAMS BY CLASS

8. Feedback

No validation suite can ever be perfect. Since its task is infinite, one can only approach perfection with varying degrees of success. Consequently, it is very important that a continuous revision of the suite is maintained, with three main objectives in mind:

(a) Removal or modification of tests that do not agree with requirements of the Pascal Standard.
(b) Addition of new tests arising from experience and context changes, and therefore unexpected interpretations.
(c) Expansion of quality control tests aimed at improving the quality of Pascal processors in all dimensions of choice.

Suggestions for categories (b) and (c) are therefore welcome. Complaints about category (a) are even more important. All will be read, digested, and carefully considered, though clearly some purely idiosyncratic tests may not make it into the package.

There is no formal bug-reporting service. Simply write to me at the following addresses:

up to January 1980

Professor A.H.J. Sale,
c/- Professor D.W. Barron,
Department of Computer Studies,
The University,
Southampton, England SO9 5NH
UNITED KINGDOM.

from January 1980

Professor A.H.J. Sale,
Department of Information Science,
University of Tasmania,
GPO Box 252C,
Hobart, Tasmania 7001
AUSTRALIA.

Please enclose a listing and any other information relative to the report/request.

But this is not enough! Simply casting the validation suite out into the wide world calls out for more. My own curiosity is reason enough, but in addition it has become apparent that overseeing the results of other people's passes of the validation suite against a processor can be extremely valuable.

Accordingly I ask you to write to me if you are prepared to send documentation on a validation run for a processor I haven't already collected. I will let you know if we already have data on that system. The documentation I would want is:

(a) a complete set of listings of runs of the validation programs, annotated to explain any obscure effects.
(b) a validation report, similar to the one produced for our own compiler.
(c) an accurate identification of the processor (compiler source and date of acquisition or version, machine identification). A manual would be useful too.

With help we can begin to assemble a comparative list of processor performance, and watch the way the situation evolves.

9. Utilization

I hope to see copies of validation reports for processors of significant interest in future editions of Pascal News. Obviously implementors will want to fix minor bugs, but may baulk at fixing difficult ones or publishing the results. Some comparative results published by responsible users will assist readers of the News to assess comparative merits of compilers, may bring some collective user pressure to bear on the implementors/maintainers to fix even the more persistent and difficult bugs, and will assist the Users Group to assess which systems are still active or most in use. I hope to see the validation suite being used by at least two groups of people: the implementors/maintainers and the users.

Implementors of course will use the package to check out their product; maintainers would be well-advised to do the same after each major revision. They will also be able to compare their efforts against other compilers more easily, and a bit more competition will be good for both suppliers and users.

Users will be able to use the package to bring pressure on suppliers to conform to the draft standard, and therefore use the package to evaluate the relative merit of two or more systems. Also, because the validation suite is virtually the Pascal Standard cast into test programs (with some reservations), access to the essential concepts of the Pascal Standard will be made more clear by inspection of the conformance and deviance tests.

Naturally, as the draft Pascal Standard stabilizes, the agreed resolutions will be embodied in further tests, thereby both providing an enforcement technique, and a method of publicizing the agreement. For example, this may happen in the area of the *pack* and *unpack* procedures which are perhaps overly restrictively defined.

10. A Confession

Once upon a time, long ago, programmers who could write tricky code were prized. This pleased them, because they were being well-rewarded for doing what they enjoyed: creating private masterpieces of complexity on a small scale. Any modern abstract artist would understand the feeling.

Now, programming has completely changed. We strive to write correct programs together with their proofs. The creativity, and the accompanying surge of pleasurable sensations on completion of a particularly difficult task, are still with us, but in a different form, and with different nuances and vibrations. I enjoy this tremendously when administrative chores allow me to indulge myself in writing programs, and I wouldn't give up the advances we've made in programming methodology for anything.

And yet, there is still that fascination with those tricky, nasty jewels we created. I must therefore confess that I believe that when the last tricky programmer on earth expires his orher last breath over a soggy sheet of paper, he or she will probably be writing a validation program. It seems to be the last refuge left for contorted thought, for to paraphrase a famous fictional detective (I think it was Hercule Poirot), how can you discover what crime has been done unless you can put yourself into the frame of mind of the criminal?

11. Some favourite tests

I must confess to having some favourite tests, which either are utterly useless and indulge a peculiar sense of humour, or are particularly devastating. I share them with you in case you might, too, share this sense of humour.

(a) Syntactic

Test 6.4.2.3-1
The interesting thing here is the peculiar scalar

```
type
    singularitytype = (me);
```

which doesn't seem to be useful for anything, though you can assign to it, test it, etc. The apparently similar (test 6.4.1-1):

```
type
    purelink = ↑ purelink;
```

is less amusing because it does have at least two distinct potential uses.

(b) Context-sensitive

Test 6.4.3.3-4
Here the use of a field-name which is already defined trips up a number of compilers. They don't wait to find the colon before rushing into analysis. Most Pascal-P compilers inherit this one.

(c) Scope

Test 6.2.2-2
The sheer perversity of being able to write

```
if true = false then writeln ('PASS')
```

is delightful.

(d) Execution

Test 6.8.3.9-7
The beauty of this one lies in two aspects: many processors completely fail it, and our compiler turned out to pass it quite unexpectedly. It comes as a surprise to devise a test that you confidently expect to fail on your own implementation, and then the thing makes a fool of you by working.... And then you have to work out how it outsmarted you. To add to its perverse charm, it usually turns out to have a simple resolution which, though unattractive, requires only that *maxint* be reduced by one.

(e) Errorhandling

Test 6.6.5.2-6
I must admit that the attraction of this one lies both in the weird variety of effects it can evoke, and to some extent by the clarity with which the aliasing problem points an accusing finger at the file buffer concept. Interestingly, a simple restriction would remove the problem, by simply not permitting file-buffers in such contexts, but it would introduce more irregularity....

Acknowledgements

I wish to acknowledge the great debt the Validation Suite owes to a multitude of people. Those I single out for special mention here contributed especially, but there are many other contributors for which there is not sufficient space.

Brian Wichmann : for initiating the project, for carrying it out throughout the first phase, and for many insights and tests, and as joint author.

Andy Mickel : for encouragement to continue, despite over-runs.

R.D. Tennent : for many critical comments on semantics.

Roy Freak : for patient and hard work in assembling around 300 programs to a consistent style and putting up with my nit-picking and niggling.

Nigel Saville : for diligence in interpreting often partial instructions and creating a large number of provably correct (or provably incorrect) programs in the rewriting phase.

Jenny Mizielinski : for laboriously and carefully typing (seemingly endlessly) abtruse documents full of mysterious numbers, each of which was highly significant.

The Sale Family : for putting up with grunts and groans, and with listings strewn around the sitting room.

Of course, to all my correspondents who contributed unknowingly must go a special kind of thanks. Without your stimulation, the Validation Suite might not have had the firm basis it now has.

Arthur Sale
Tasmania

1979 August

THE PASCAL VALIDATION SUITE
Version 2.2

DISTRIBUTION INFORMATION AS AT 20 AUGUST 1979

Revision History

Version 2.0 was the first release of a completely rewritten package which was based on earlier work by B.A. Wichmann and A.H.J. Sale. This earlier work is considered to be version 1, and is now obsolete.

Version 2.0 was indexed to Working Draft 3 of the Pascal Standard as published in Pascal News #14. Version 2.1 fixed up a few bugs detected after release. Version 2.2 is altered in indexing to refer to the draft ISO Standard document ISO/TC97/SC5/N462, and incorporates more tests which facilitate the timing and measurement of quality in Pascal processors. Subsequent revisions will be issued when either detected errors in the package require a revision, or when a new version of the draft standard is released.

Purpose

The validation suite is provided to exercise Pascal processors and to determine by testing whether the processor conforms to the requirements of the Pascal Standard or not, and to provide a common set of programs for documenting implementation-dependencies and quality of implementation. It is strongly oriented around the draft Pascal Standard, and tests are suggested by that document. A few proposed tests have been omitted because it has been suggested that the draft Standard will be revised in that area, but these are few. It follows therefore that any revisions of the draft Standard will cause revisions of the validation suite. The suite currently contains over 300 programs.

Acquisition

To acquire a machine-readable copy of the validation suite, apply to one of the distribution centres. It will be necessary to fill in a software licence, and a small fee (around US$50) is charged to cover costs. The fee covers the supply of a magnetic tape, the copying of the validation suite onto the tape, airmail postage to the address provided, and a limited notification service relating to later releases or inaccuracies detected in the suite.

Restrictions

The conditions of release prohibit the distribution of the package to third parties so as to limit the growth of unauthorized and inaccurate versions. The likely incidence of change if the draft standard is revised will show the desirability of this requirement. However, no restriction is placed on the use of the package for validating Pascal processors, for benchmarking, for acceptance tests, for preparing comparative reports, and similar activities, nor on the distribution of the results of such use.

The validation suite is expected to be widely used and distributed, not restricted to a small subset of the user community.

Feedback

No special reporting mechanisms have been set up. However, the authors will attempt to produce revisions of the validation suite and distribute them to the distribution centres as necessary, and to publicize their availability.

Information relating to a pass of the validation programs against a particular processor would be welcome at the University of Tasmania, but it must be understood that the authors cannot provide full reports on all listings provided.

Of particular value would be any tests that produced entirely unexpected results which may be of wider interest, or which are without any reasonable explanation from the user's point of view. In some cases the authors may be able to deduce the likely cause, or recognize an epidemic of common flaws. Any correspondance which points out an error in the classification of the programs in the suite, or in its construction, or suggests a new test, would be most welcome.

Address for suggestions or complaints:

September '79 - January '80:

Professor A.H.J. Sale,
c/- Department of Computer Studies,
The University,
Southampton, England SO9 5NH
UNITED KINGDOM.

February '80 onwards:

Professor A.H.J. Sale,
Department of Information Science,
University of Tasmania,
GPO Box 252C,
Hobart, Tasmania 7001
AUSTRALIA.

Distribution Format and Addresses

```
Character set

 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~

THIS TAPE CONTAINS 4 FILES :

FILE 1:
     THE CHARACTER SET USED IN THE VALIDATION SUITE AND DETAILS
     OF THE TAPE STRUCTURE.

FILE 2:
     THE SKELETON PROGRAM WHICH IDENTIFIES EACH TEST PROGRAM

FILE 3:
     THE DOCUMENTATION OF THE PASCAL PROCESSOR VALIDATION SUITE

FILE 4:
     THE SUITE OF PASCAL TEST PROGRAMS

EACH FILE CONSISTS OF 80 CHARACTER RECORDS.
A SEQUENCE NUMBER IS IN COLUMNS 73-80 INCLUSIVE.
FILEMARKS SEPARATE EACH FILE.
TWO FILEMARKS AT END OF TAPE.
THE TAPE IS UNLABELLED.

TAPES SUPPLIED BY THE UNIVERSITY OF TASMANIA AND DR B.A. WICHMANN
HAVE THE FOLLOWING FORMAT:

  15 RECORDS PER BLOCK.
  EACH RECORD IS WRITTEN IN ASCII.
  THE TAPE WILL NORMALLY BE WRITTEN AT 1600BPI, PE TAPE.

TAPES SUPPLIED BY R.J. CICHELLI ARE WRITTEN AT
  800 BPI, NRZ, ODD PARITY, 9-TRACK
  AND 1) EITHER ASCII OR EBCDIC
      2) 1,10,20 RECORDS PER BLOCK

IN EITHER CASE, OTHER TAPE SPECIFICATIONS MAY BE ARRANGED BY CONTACTING
THE DISTRIBUTERS.

FILES 2, 3 AND 4 CONTAIN BOTH UPPER AND LOWER CASE LETTERS.
FILE 1 CONTAINS UPPER CASE LETTERS ONLY EXCEPT FOR THE SPECIFICATION
OF THE CHARACTER SET.

VERSION 2.2

JULY 1979
```

```
ADDRESSES:
  DISTRIBUTION IN AUSTRALIA, NEW ZEALAND AND JAPAN.
  ALSO ANY GENERAL CORRESPONDENCE, CORRECTIONS, QUERIES.

    PASCAL SUPPORT,
    DEPARTMENT OF INFORMATION SCIENCE,
    UNIVERSITY OF TASMANIA,
    BOX 252C, G.P.O.,
    HOBART 7000
    TASMANIA
    AUSTRALIA.
    61-02-230561 EXT 435

    DISTRIBUTION COST: $A50

  FOR DISTRIBUTION IN NORTH AMERICA -

    RICHARD J. CICHELLI,
    C/- ANPA RESEARCH INSTITUTE
    BOX 598
    EASTON, PA. 18042
    U.S.A.
    (215) 253-6155

    DISTRIBUTION COST: $US50

  FOR DISTRIBUTION IN EUROPE -

    DR. B. WICHMANN,
    NATIONAL PHYSICS LABORATORY,
    TEDDINGTON, TW11 0LW
    MIDDLESEX,
    ENGLAND
    01-977 3222 EXT 3976

    DISTRIBUTION COST: NOT KNOWN
```

DEPARTMENT OF INFORMATION SCIENCE
UNIVERSITY OF TASMANIA
BOX 252C, G.P.O.,
HOBART 7001.
TASMANIA
AUSTRALIA

A Pascal Processor Validation Suite

by B.A. Wichmann
National Physical Laboratory
Teddington, Middlesex,
TW11 0LW, United Kingdom

A.H.J. Sale
Department of Information Science
University of Tasmania,
GPO Box 252C,
HOBART, Tasmania 7001

Abstract

The document describes a series of test programs written in Pascal. The suite of programs may be used to validate a Pascal processor by presenting it with a series of programs which it should, or should not, accept. The suite also contains a number of programs that explore implementation-defined features and the quality of the processor. The tests are generally based on the draft ISO Standard for Pascal.

NOTE

This is a working document. It is being continually revised and extended. Comments, corrections, extra tests, and results of running any tests would be most welcome.

Dated: 20 August, 1979

Version: 2.2

1. INTRODUCTION AND PURPOSE

This paper describes a suite of test programs which has been designed to support the draft Standard (Addyman, 1979) for the programming language Pascal (Jensen & Wirth, 1975) prepared for approval by ISO. (In the rest of this paper, the draft Standard is simply referred to as the Standard). It therefore follows similar work done by AFSC (1970) for COBOL, and Wichmann (1973,76,77) for Algol 60.

The suite of programs is called a *validation suite* for Pascal processors; however it is important to emphasize that no amount of testing can assure that a processor that passes all tests is error-free. Inherent in each test are some assumptions about possible processors and their designs; a processor which violates an assumption may apparently pass the test without doing so in reality. Also, some violations may simply not be tested because they never occurred to the validation suite designers, nor were generated from the draft Standard.

Two examples may illustrate this as a warning to users against expecting too much. Firstly, consider a fully interpretable Pascal processor. It may pass a test which contains a declaration which it would mis-handle otherwise, simply because the program did not include an access to the object concerned so that it was never interpreted. A second example might be a Pascal processor which employs a transformation of the Pascal syntax rules. Since the pathological cases incorporated into the test programs are based on the original rules, a mistake in transformation may not be detected by the test programs.

On the other hand, the test series contains a large number of test cases which exercise a Pascal processor fairly thoroughly. Hence passing the tests is a strong indication that the processor is well-designed and unlikely to give trouble in use. The validation suite may therefore be of interest to two main groups: implementors of Pascal, and users of Pascal.

Implementors of Pascal may use the test series to assist them in producing an error-free processor. The large number of tests, and their independent origin, will assist in detecting many probable implementation errors. The series may also be of use for re-validation of a processor after modification to incorporate a new feature, or to fix an error.

Users of Pascal, which includes actual programming users, users of Pascal-written software, prospective purchasers of Pascal processors, and many others, will also be interested in the validation suite. For them it will provide an opportunity to measure the quality of an implementation, and to bring pressure on implementors to provide a correct implementation of Standard Pascal. In turn, this will improve the portability of Pascal programs. To emphasize this role, the validation suite also contains some programs which explore features which are permitted to be implementation defined, and some tests which seek to make quality judgements on the processor. The validation suite is therefore an important weapon for users to use in influencing suppliers.

Naturally, implementors of Pascal are best placed to understand why a processor fails a particular test, and how to remedy the fault. However, the users' view of a Pascal processor is mainly at the Pascal language level, and the fact of a failure is sufficient for the users' purpose.

2. THE TEST PROGRAM STRUCTURE

Each test program follows a consistent structure to aid users of the suite in handling them. Most of the following rules apply to all programs: a few hold everywhere except in a few test cases meant to test the particular feature involved. Such rules are marked by an asterisk, and a following note points out the exceptions.

(i) Each program starts with a *header comment*, whose structure is given later.

(ii) The header comment is always immediately followed by an explanatory comment in plain English, which describes the test to be carried out and its probable results.

(iii) Each program closes with the characters "end." in the first four character positions of a line. This pattern does not otherwise occur in the program text.

(iv) All program lines are limited to 72 character positions.

(v)* The lexical tokens used are in conformance with the conventions set out in the draft ISO standard, and reproduced in an appendix. Thus comments are enclosed in curly brackets, the not-equal token is "<>", etc. In addition, all program text is in lower case letters, with mixed-case used in comments in accordance with normal English usage. String- and character-constants are always given in upper-case letters. (Note: A few tests set out to check lexical handling, and may violate these rules. Translation of mixed cases to one case will therefore make these tests irrelevant, but will have no other effect.)

(vi) Direct textual replacement of any lexical token, or the comment markers, with the approved equivalents given in the Standard, will not cause the significant text on a program line to exceed 72 characters.

(vii)* The program writes to the default file output, which is therefore declared in the program heading. (Note: one test - the minimal program - does no printing; a few cross-references are virtually the same).

2.1 The header comment

The header comment always begins with the characters "{TEST" in positions 1-5 of a line. No other comments are permitted to have the character "{" and "T" directly juxtaposed in this way. The syntax of a header comment in EBNF is given by:

```
header-comment  =  "{TEST" program-number "," "CLASS=" category-name "}"  .

program-number  =  number  {"." number}  "-" number  .

number          =  digit { digit}  .

category-name   =  "CONFORMANCE"|"DEVIANCE"|"IMPLEMENTATIONDEFINED"|
                   "ERRORHANDLING"|"QUALITY"|"EXTENSION"  .
```

For example, a possible header comment is:

```
{TEST 6.5.3-10, CLASS=CONFORMANCE}
```

The program number identifies a section in the Standard which gives rise to the test, and a serial number following the dash to uniquely identify each test within that section. If other sections of the Standard are relevant, the explanatory comment will mention them. The program title is constructed from the section number by replacing "TEST" by "t", "." by "p" for point, and "-" by "d" for dash. Thus the above header comment belongs to a program *t6p5p3d10*. This technique may also be used to name a program source text file name in processing.

The category-name identifies a class into which this test falls. The function and design of each test depends on its class. These are explained later. Thus it is possible to read through the validation suite file and simply identify the header comment by the leading "{T" in the first two character positions, identify its section relevance and construct a unique identifier for each program, and to select programs of particular classes.

2.2 The program classes

2.2.1 *CLASS=CONFORMANCE*

The simplest category to explain is CLASS=CONFORMANCE. These programs are always correct standard Pascal, and should compile and execute. With one exception (the minimal program), the program should print "PASS" and the test number if the program behaves as expected. In some cases an erroneous interpretation causes the program to print "FAIL"; in other cases it may fail before doing this (in execution, loading, or at compilation). Conformance tests are derived directly from the requirements of the Standard, and attempt to ensure that processors do indeed provide the features that the Standard says are part of Pascal, and that they behave as defined. Since conforming programs execute to completion, typical conformance tests will include a number of related features; all will be exercised by processors that pass.

2.2.2 *CLASS=DEVIANCE*

The next simplest category is CLASS=DEVIANCE. These programs are never standard Pascal, but differ from it in some subtle way. They serve to detect processors that meet one or more of the following criteria:

(a) the processor handles an extension of Pascal,
(b) the processor fails to check or limit some Pascal feature appropriately, or
(c) the processor incorporates some common error.

Ideally, a processor should report clearly on all deviance tests that they are extensions, or programming errors. This report should be at compile-time if possible, or in some cases in execution. A processor does not conform to the Standard if it executes to completion. In such cases the program will print a message beginning "DEVIATES", and users of the tests must distinguish between genuine extensions and errors. (In a few cases a possible extension is tested also for consistency under this class.)

It is obviously not possible to test all possible errors or extensions. The deviance tests are therefore generated from some assumptions about implementation (which may differ from test to test), and from experience with past flaws detected. No attempt is made to detect extensions based on new statement types or procedures, but attention is concentrated on more stable areas. Obviously since each deviance test is oriented to one feature, they tend to be shorter than conformance tests, and to generate a short series where one conformance test collects several examples.

2.2.3 *CLASS=IMPLEMENTATIONDEFINED*

In some sections of the Standard, implementors are permitted to exercise some freedom in implementing a feature. An example is the significance limit of identifiers; another is the evaluation order of boolean expressions. The CLASS=IMPLEMENTATIONDEFINED tests are designed to report on the handling of such features. A processor may fail these tests by not handling them correctly, but generally should execute and print some message detailing the implementation dependency. The collection of such implementation dependencies is useful to the writers of portable software. Some tests in this category require care in interpretation, as the messages generated by the test program rely on some assumptions about the processor implementation. The programs may or may not be standard Pascal: often they are not.

For example, one program attempts to measure the significance limits of identifiers by declaring a series of differing length in an inner procedure that are different from an outer series by their last letter. Thus it violates the requirement for uniqueness over the first eight characters and relies on masquerading redefinition under the scope rules for its effect. One processor, however, reports that just this is happening during compilation. Though this is ideal behaviour, it would destroy the test if the program then was not permitted to run. (In this case, in fact, the messages were only warnings.)

2.2.4 *CLASS=ERRORHANDLING*

The Standard specifies a number of situations by stating that "an error occurs if" the situation occurs. The tests of this class evoke one (and only one) such error. They are therefore not in Standard Pascal with respect to this feature, but otherwise conform.

A correct processor will detect each error, most probably as it occurs during execution but possibly at an earlier time, and would give some explicit indication of the error to the user. Processors that fail to detect the error will exhibit some undefined behaviour: the tests enable these cases to be identified, and allows for documentation of the handling of detected errors.

2.2.5 *CLASS=QUALITY*

These tests are a miscellany of test programs which have as their only common feature that they explore in some sense the quality of an implementation. The tests include the following, amongst others:

* tests that can be timed, or used to estimate the performance.
* tests that have known syntax errors which can be used to inspect the diagnostics.

* tests that establish whether the implementation has a limit which is a virtual infinity in some list or recursive production. For example a deep nesting of for-loops (but not unreasonable!) would see whether there was any limit, perhaps due to a shortage of registers on a computer.

2.2.6 *CLASS=EXTENSION*

A final category is CLASS=EXTENSION. These are specific to some conventionalized extension approved by the Pascal Users Group, such as the provision of an otherwise clause in case statements. In this case, the class in the header comment is followed by a sub-class, as in the example:

{TEST 6.8-1, CLASS=EXTENSION, SUBCLASS=CONFORMANCE}

The subclass gives the purpose of the test according to the previously explained classes.

3. STRUCTURE OF THE VALIDATION SUITE

The validation suite as distributed consists of:

A. Machine-readable files
   1. A header file containing the character set and an explanation of the structure of the other files.
   2. A skeleton program, written in Pascal, to operate on the final file of tests.
   3. A copy of this document in machine readable form.
   4. A file consisting of the sequence of test programs arranged in lexicographic order of their program-number (see section 2.1).

B. Printed materials
   1. This document.
   2. A printed version of A.1.

The skeleton program as supplied prints the test programs on the output file, but calls a procedure *newprogram* before listing the start of a program, and calls a procedure *endprogram* after printing the last end of a program. These procedures as now supplied simply print a heading and a separator respectively. However, users of the suite may write versions of *newprogram* and *endprogram* that may write programs to different named files, and which may initiate jobs in the operating system queues to carry out the tests. The two procedures *newsuite* and *endsuite* are also provided in case these are of use.

Since *newprogram* may return a status result, it may also be programmed to be selective in its handling of tests. Only conformance tests may be selected, or only tests in section 6.3, as required.

The skeleton program is in standard Pascal, and conforms to the conventions of the validation suite (but has no header comment). It is documented in an Appendix.

4. REPORTING THE RESULTS

The results of a pass of the validation suite against a Pascal processor should be reported in a standard way, illustrated by the schema below.

PASCAL PROCESSOR IDENTIFICATION (host, computer, origin of processor, version):
TEST CONDITIONS (tester, date, test version):
CONFORMANCE TESTS
Number of tests passed = ?
Number of tests failed = ?
    Details of failed tests:
    TEST ???? : explanation of why or what
    .....
DEVIANCE TESTS
Number of deviations correctly detected = ?
Number of tests showing true extensions = ?
Number of tests not detecting erroneous deviations = ?
    Details of extension:
    .....
    Details of deviations:
    .....
ERROR-HANDLING
Number of errors correctly detected = ?
Number of errors not detected = ?
    Details of errors not detected:
    .....
IMPLEMENTATION DEFINED
Number of tests run = ?
Number of tests incorrectly handled = ?
    Details of implementation-dependence:
    .....
QUALITY MEASUREMENT
Number of tests run = ?
Number of tests incorrectly handled = ?
    Results of tests:
    .....
EXTENSIONS
Number of tests run = ?
Extension present = ?
    as above for this extension.

5. ACKNOWLEDGEMENTS

The authors gratefully acknowledge the assistance of many colleagues who have collected difficult cases and bugs in their compilers and have passed them on for inspirational purposes. Many of the tests have been derived from work of B.A. Wichmann (1973) and A.H.J. Sale (1978) and significant contributions have also been made by A.M. Addyman and R.D. Tennent. R. Freak and N. Saville contributed greatly by bringing consistency and care into the large effort required to assemble the validation suite itself.

The present level of the suite would not have been possible without the work of BSI DPS/13/4 in drawing up the draft ISO Standard, nor without the support of the Pascal Users Group.

REFERENCES

Addyman, A.M. (1979): BSI/ISO Working Draft of Standard Pascal by the BSI DPS/13/4 Working Group, Pascal News No 14, January 1979, pp4-54. See also ISO/TC97/SC5/N492

AFSC (1970): *User's Manual, COBOL Compiler Validation System*, Hancom Field, Mass., 1970.

DeMorgan, R.M., Hill, I.D., Wichmann, B.A. (1977): *Modified Report on the Algorithmic Language ALGOL 60*, Comp. J. Vol 19 No 4. pp364-379. 1977

Sale, A.H.J. (1978): *Pascal Compatibility Report (Revision 2)*, Department of Information Science Report R78-3, University of Tasmania, May 1978. *{obsoleted by this document}*

Wichmann, B.A. (1973): *Some Validation tests for an Algol compiler*, NPL Report NAC 33, March 1973.

Wichmann, B.A. & Jones, B. (1976): *Testing ALGOL 60 Compilers*, Software - Practice and Experience, Vol 6 pp261-270. 1976

Appendix : Documentation of Skeleton Program

Purpose

The skeleton program provided is a standard-conforming Pascal program that will identify each test in a file of test programs. It is distributed with five stubs which are intended for user modification so that the program will serve as the parent of an automatic system for running the tests. With the large number of tests in the validation suite, such a system is important. As distributed the stubs simply print the text of the test program.

Implementation

If the recipient's Pascal processor will accept upper- and lower-case letters, and the ISO standard lexical representation, then the program should be able to run directly. If lexical substitutions are necessary, these should be made. No lines approach the limiting length of 72 closely, so some expansion room is possible. If only one case of letters is available for the source text, then the whole program should be upper-cased with the exception of the character constants in the main program and the procedure *convert*. If however, the Pascal processor is also limited to upper-case characters in the *char* type, these too will have to be converted, and the whole of the validation suite will have to be converted to upper-case alone before the program will run.

Modification

Once the recipient has verified that he has the skeleton working correctly, it can be modified to other purposes. If the only use desired is the construction of many individual files, each with a single test program, then *newsuite* and *endsuite* may be made dummy procedures. Procedure *newprogram* can open (rewrite) a file with the converted name supplied, which is guaranteed unique, leaving *processline* to write the text to this file and *endprogram* to do any necessary closing of the file.

Alternatively, the program may be modified to construct a job deck including job control statements. The *newprogram* procedure will have some more complex actions to take, and the *endprogram* procedure will initiate the job into the operating system queues. In the event that the user wants to batch up the tests in lots of 20 or so, the access to the variable *count* allows *newprogram* and *endprogram* to take appropriate action every twentieth test. Or again, if the whole lot is to be batched or submitted as a simulated time-sharing job, the *newsuite* and *endsuite* programs can be used to initialize and initiate.

Additionally, the *newprogram* procedure is given access to the test program name and its class. Specially tailored programs can be written to only initiate conformance tests, or only conformance tests relevant to section 6.5 of the Standard, or whatever the user desires. If necessary, specifications could be read in, most easily by a modified version of the *newsuite* procedure.

System documentation

The main program alternates between dormant states where it searches for the header comment starting a test, and active states where it processes lines keeping an eye out for the closing end of the test. When it finds a header comment, it extracts the data for passing to the stub procedures *newprogram* and *endprogram*.

The procedure *extract* does this extraction from the line buffer. It essentially assumes correctness of the header syntax. The program name is stored in a special record.

The procedure *convert* is not necessary to the distributed version apart from a demonstrative use. It is provided to convert test program numbers from numeric format (eg. 6.2.8-4) to an alphabetic format suitable as an identifier or file name (e.g. t6p2p8d4). It switches the format in the name record from whatever it is to the alternative format. The format as supplied to *newprogram* is digitized.

The procedure *readaline* does simply that. The line is stored in an internal buffer *line*.

The procedures *newprogram* and *endprogram* have value parameters with exactly the same name as their global counterparts. This is to hide the global variables from them and provide some measure of robustness against errors. The *status* parameter of *newprogram* is listed explicitly to emphasize the possible change of value.

There are few features of Pascal in the program that may not be implemented. There are two goto statements leading to a disaster-exit label; if non-local gotos are not implemented the gotos may be omitted or replaced by a *halt* equivalent. The packing in *charvectype* is only provided to allow comparisons with character array constants ("strings"). All identifiers have been checked for uniqueness over the first eight characters, and no non-standard usages have been detected in the program. If it is necessary to alter the program to recognize header comments by '(*T' instead of '{T', some changes will be necessary on two lines of the main program.

Lexical structure of procedures†

```
main
   readaline
   convert
   extract
      scan
   newsuite*
   endsuite*
   newprogram*
   endprogram*
   processline*
```

Notes:

* user-modifiable stub procedures.

† a call structure diagram is also provided overleaf.

Distributed version

As distributed the skeleton program prints all the test programs, one per page, in sequence. The headings simply illustrate some features of the package. Warning: this takes a lot of paper, so be prepared.

Date of documentation

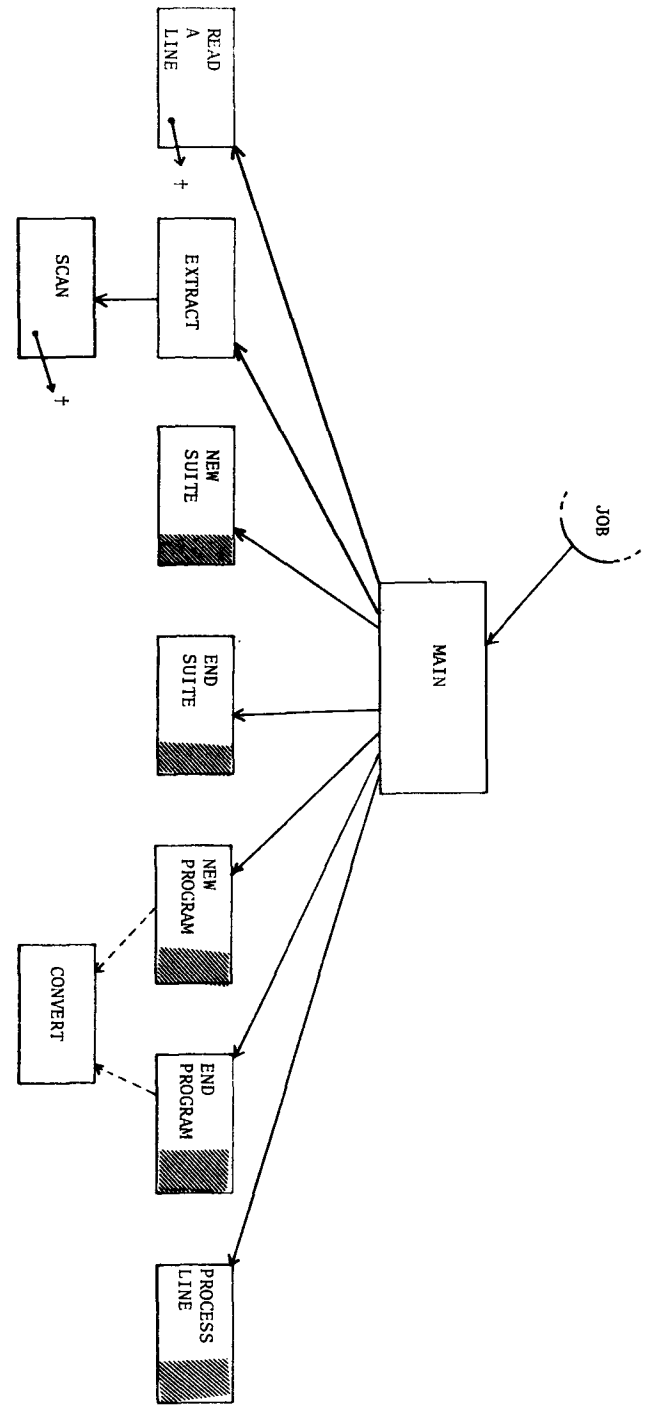
1979 July 13

Author

A.H.J. Sale, University of Tasmania

Call Structure Diagram for procedures

{†goto 999 in main program}



```
{
*************************************************************************
*                                                                       *
* This skeleton program is provided to process the file of test         *
* programs in the validation suite.  As supplied, it simply lists       *
* the test programs in a suitable format, but it is provided with       *
* stubs which can be modified by users of the suite to select           *
* individual test programs, classes of programs, or particular          *
* sub-classes, and to write these to named files or initiate them       *
* in a job stream.                                                      *
*                                                                       *
* The particular stubs of interest are:                                 *
*     newsuite                                                          *
*        newprogram                                                     *
*           processline                                                 *
*        endprogram                                                     *
*     endsuite                                                          *
* As supplied these are practically only dummy routines.                *
*                                                                       *
* This program is written in Standard Pascal according to the           *
* ISO Standard.  It should compile and run without error if             *
* its lexical representation is acceptable to your processor.           *
*                                                                       *
* (C) Copyright 1979 A.H.J.Sale, University of Tasmania.                *
*                                                                       *
*************************************************************************
 }

program skeleton(input,output);

label 999; { used for disaster exits }

const

   maxnamesize=         20;
                        { size of field in name record }
   nameoverflow=        21;
                        { maxnamesize + 1 }
   maxlinesize=         72;
                        { size of line array }
   lineoverflow=        73;
                        { maxlinesize + 1 }

type

   namesize=            1..maxnamesize;

   linesize=            1..maxlinesize;

   statustype=          (dormant,active,terminated);
                        { Is program being processed? }

   charvectype=         packed array[namesize] of char;
                        { names }
```



```
   nametype=            record
                           charvec:    charvectype;
                           length:     0..maxnamesize;
                           version:    (digitized,alphabetized)
                        end;
                        { used to hold program names }

   natural=             0..maxint;
                        { very common type }

   linetype=            array[linesize] of char;
                        { used for line buffers }

   loopcontrol=         (scanning,found,notfound);
                        { for controlling scan loops }

   classtype=           (conformance,deviance,implementationdefined,
                           errorhandling,quality,other);
                        { category of test }

var

   name:                nametype;
                        { name of current program }

   class:               classtype;
                        { category of current program }

   status:              statustype;
                        { current status }

   count:               natural;
                        { program sequence number }

   line:                linetype;
                        { the line buffer }

   linelength:          0..maxlinesize;
                        { holds actual number of chars in line }
```

```
procedure readaline;

{ Reads an input line }

var
   i :   0..lineoverflow;
   ch:   char;
begin
   if eof(input) then begin
      writeln(output);
      writeln(output,' ****ERROR IN READALINE - HIT EOF');
      goto 999
   end else begin
      i:=0;
      while not eoln(input) do begin
         i:=i+1;
         if (i > maxlinesize) then begin
            writeln(output);
            writeln(output,' ****ERROR IN READALINE, LONG LINE');
            goto 999
         end;
         read(input,line[i])
      end;
      read(input,ch); { disposing of the line marker }
      linelength:=i
      { Textfiles must have eoln before eof }
   end
end; { of procedure readaline }
```



```
procedure convert(var name:nametype);

{ This procedure exchanges the representation of the name
  between digitized (eg 6.2-1) and alphabetized (eg t6p2d1).
  It inspects the version code and reverses it. }

var
   i :   linesize;
   ch:   char;
begin

   if (name.version = digitized) then begin
      { We need to alphabetize the name }
      { Extract assures that digitized length never exceeds
        maxnamesize-1, so leaving one char expansion space. }
      for i:= name.length downto 1 do begin
         case name.charvec[i] of
            '.': ch:='p';
            '-': ch:='d';
            '0','1','2','3','4','5','6','7','8','9':
                  ch:=name.charvec[i]
         end;
         name.charvec[i+1]:=ch
      end;
      name.charvec[1]:='t';
      { Name is converted }
      name.length:=name.length+1;
      name.version:=alphabetized

   end else begin { version = alphabetized }
      { We need to digitize the name }
      for i:=2 to name.length do begin
         case name.charvec[i] of
            'p': ch:='.';
            'd': ch:='-';
            '0','1','2','3','4','5','6','7','8','9':
              ch:=name.charvec[i]
         end;
         name.charvec[i-1]:=ch
      end;
      name.charvec[name.length]:=' ';
      { Name is converted }
      name.length:=name.length-1;
      name.version:=digitized

   end { of if }
end; { of procedure convert }
```

```
procedure extract;

{ This procedure extracts the program-number and class from the
  header comment of a test program.  It sets global parameters
  name and class.  There is some simple error-handling in the
  procedure, but it is generally assumed that the header
  comments conform to the syntax.  The error-handling is just
  in case. }

var
   i,                      { used to scan the line }
   isave: 1..lineoverflow; { used to save value of i }
   lengthofname,           { holds length of name found }
   k:     namesize;        { used to fill name }


   procedure scan(lowch,highch:char);

   { Scan moves the index i along the line until it finds
     a character lying between lowch and highch inclusive.
     It includes some simple error-handling which terminates
     the program. }

   var
      state : loopcontrol;

   begin
      { Set loop to scan forwards }
      state:=scanning;
      { Loop invariant R1 =
        "characters from line[initial i] to line[i-1] are
         not in the desired subrange." }
      while (state = scanning) do begin
         if (i > linelength) then begin
            { No more to go, so get out }
            state:=notfound
         end else if (line[i] >= lowch) and (line[i] <= highch)
            then begin
            state:=found
         end else begin { Char is not in range }
            i:=i+1
         end { of if }
      end; { of while }

      if (state = notfound) then begin
         writeln(output);
         writeln(output,' ****ERROR IN SCAN - REACHED LINE END');
         goto 999  { in outer block and give up }
      end;

      { Return, leaving i at found character }

   end; { of procedure scan }
```



```
begin { of extract }

   { Start scanning at 1 even though we know it begins '{T' }
   i:=1;
   { Scan until we find a digit }
   scan('0','9');
   { Now assured of a digit so save the index, and look for
     the closing comma }
   isave:=i;
   scan(',',',');
   { In case spaces between number and comma }
   while (line[i-1] = ' ') do i:=i-1;
   { Fill in the discovered name }
   lengthofname:=i-isave;
   if (lengthofname >= maxnamesize) then begin
      lengthofname:=maxnamesize-1
   end;
   for k:=1 to lengthofname do begin
      name.charvec[k]:=line[isave+k-1]
   end;
   { Space-fill so as to allow string comparisons }
   for k:=(lengthofname+1) to maxnamesize do begin
      name.charvec[k]:=' '
   end;
   { And fill in the rest of the record }
   name.length:=lengthofname;
   name.version:=digitized;

   { Now scan for the '=' sign that precedes the class }
   scan('=','=');
   { And an alphabetic character following. The test may
     also let through some non-alphabetics, but no matter. }
   scan('A','Z');
   { Identify the class by its first letter.  It is always
     upper-case. }
   if (line[i] = 'C') then class:=conformance
   else if (line[i] = 'D') then class:=deviance
   else if (line[i] = 'E') then class:=errorhandling
   else if (line[i] = 'I') then class:=implementationdefined
   else if (line[i] = 'Q') then class:=quality
   else begin
      { Error, not recognized }
      writeln(output);
      writeln(output,' ****ERROR IN EXTRACT - WHAT CLASS?');
      class:=other    { default }
   end;

   { Now we have established the desired values, so return }

end; { of procedure extract }
```

```
procedure newsuite;

{ This procedure may be used to initialize the run somehow, or to
  read in some parameters for the process. }

begin
   writeln(output,' THIS IS AN EXECUTION OF THE SKELETON ',
      'VALIDATION SUITE PROCESSOR');
   writeln(output,' ----------------------------------',
      '-------------------------');
   page(output)
end; { of procedure newsuite }



procedure endsuite;

{ This procedure may be used to initiate a global job, or
  to check correct completion. }

begin
   writeln(output,' SKELETON VALIDATION SUITE PROCESSOR END',
      ' (',count:4,' TESTS READ)')
end; { of procedure endsuite }
```



```
procedure newprogram(name :      nametype;
                     class:      classtype;
                     count:      natural;
                 var status:     statustype);

{ This procedure is called at the recognition of a header
  comment while in dormant status.  The first two parameters
  are derived from the header comment, while the third is
  simply the ordinal number of the test met in processing.
  The final parameter is the status of the search.

  The user's version has the responsibility of deciding what
  to do about this program by setting status to active or
  leaving it passive.  In the first case all lines are processed
  by processline later, and the user may set up any headers,
  JCL statements, etc, beforehand.  In the latter case the
  driver resumes searching for a header comment.

  The name may be in digitized or alphabetized version - see
  procedure convert.  Searching for a particular program can
  be done with a string comparison, for example:
      name.charvec = '6.2-1                '
  Searching for a subsection can be done with string comparisons
  if care is taken with collating sequence.  On ASCII machines,
  space collates lower than anything else, so that
      (name.charvec >= '6.2                ') and
      (name.charvec <  '6.3                ')
  will determine all tests relevant to section 6.2 and its
  subsections.

  The class may also be used in selection.

  The count may be used to parcel up say 20 tests and run them
  in a batch, rather than the whole shebang at one go.  An
  appropriate test is
      if ((count mod 20) = 0) then ...
  although this may be more appropriately used in endprogram.
}
var
   i :   linesize;
begin
   write(output,' TEST PROGRAM ');
   if (name.version = alphabetized) then convert(name);
   for i:=1 to name.length do write(output,name.charvec[i]);
   write(output,' (ALIAS ');
   convert(name);
   for i:=1 to name.length do write(output,name.charvec[i]);
   writeln(output,'),  NO',count:4);
   writeln(output);
   writeln(output);

   status:=active { forcing print of all }

end; { of procedure newprogram }
```

```
procedure endprogram(name:    nametype;
                     class:   classtype;
                     count:   natural);

{ See the comments for newprogram.
  Endprogram can do exactly the same tests, but it has no
  responsibility for status which will automatically
  become dormant afterwards. }

begin
   page(output)
end; { of procedure endprogram }



procedure processline;

{ This procedure processes a source line of text, whatever
  that implies.  Here we just print it. }

var
   i :   linesize;
begin
   write(output,'      ');
   for i:=1 to linelength do write(output,line[i]);
   writeln(output)
end; { of procedure processline }
```



```
begin { of Main Program }

   count:=0;
   status:=dormant;

   newsuite;          { Call in case user needs prologue }

   repeat begin { until status = terminated }
      readaline;
      if (status = dormant) then begin
         { We only look for header comments }
         if (linelength >= 2) then begin
            if (line[1] = '{') and (line[2] = 'T') then begin
               extract;
               if (name.charvec = '999                ') then begin
                  status:=terminated
               end else begin
                  count:=count+1;
                  { Newprogram may alter status too }
                  newprogram(name,class,count,status)
               end
            end
         end
      end; { having possibly processed a header }
      { If dormant we won't do anything now }
      if (status = active) then begin
         processline;
         if (linelength >= 4) then begin
            if (line[1] = 'e') and
               (line[2] = 'n') and
               (line[3] = 'd') and
               (line[4] = '.') then begin
               endprogram(name,class,count);
               status:=dormant
            end
         end
      end
   end until (status = terminated);

   endsuite;          { Call in case user needs epilogue }

999:
end.
```

```
{TEST 5.2.2-1, CLASS=QUALITY}                                          00000100
{                                                                      00000200
-------------------------------------------------------------------------00000300
                                                                       00000400
(C) Copyright                                                          00000500
                                                                       00000600
A.H.J. Sale                                                            00000700
R.A. Freak                                                             00000800
                                                                       00000900
July 1979                                                              00001000
                                                                       00001100
All rights reserved                                                    00001200
                                                                       00001300
This material may not be reproduced or copied in                       00001400
whole or part without written permission from the authors.             00001500
                                                                       00001600
Department of Information Science                                      00001700
University of Tasmania                                                 00001800
Box 252C, G.P.O.,                                                      00001900
Hobart 7000.                                                           00002000
Tasmania                                                               00002100
Australia.                                                             00002200
                                                                       00002300
-------------------------------------------------------------------------00002400
                                                                       00002500
}                                                                      00002600
                                                                       00002700
{ This program does not conform to the standard because its            00002800
  meaning is altered by the truncation of its identifiers to 8         00002900
  characters. Does the processor provide any indication that the       00003000
  program does not conform?                                            00003100
  Such surreptitious changes of meaning are dangerous.                 00003200
  Obviously processors with 8-character significance will have         00003300
  difficulty in detecting such problems, but it can be done.           00003400
  For processors with full significance it is easier. }                00003500
                                                                       00003600
program t5p2p2d1(output);                                              00003700
const                                                                  00003800
   valueofaverylongidentifier1 = 10;                                   00003900
procedure p;                                                           00004000
var                                                                    00004100
   valueofaverylongidentifier2:integer;                                00004200
begin                                                                  00004300
   valueofaverylongidentifier2:=11;                                    00004400
   if valueofaverylongidentifier1 <>                                   00004500
      valueofaverylongidentifier2 then                                 00004600
      writeln(' IDENTIFIERS DISTINGUISHED...5.2.2-1')                  00004700
   else                                                                00004800
      writeln(' IDENTIFIERS NOT DISTINGUISHED...5.2.2-1')              00004900
end;                                                                   00005000
                                                                       00005100
begin                                                                  00005200
   p                                                                   00005300
end.                                                                   00005400
```

```
{TEST 6.1.2-1, CLASS=DEVIANCE}                                         00005500
                                                                       00005600
{ This test checks that nil is implemented as a reserved               00005700
  word, as it should be. The compiler deviates if the program compiles 00005800
  and prints DEVIATES. }                                               00005900
                                                                       00006000
program t6p1p2d1(output);                                              00006100
var                                                                    00006200
   i:(tick,cross,nil);                                                 00006300
begin                                                                  00006400
   i:=nil;                                                             00006500
   writeln(' DEVIATES...6.1.2-1, NIL')                                 00006600
end.                                                                   00006700


{TEST 6.1.2-2, CLASS=DEVIANCE}                                         00006800
                                                                       00006900
{ This test checks that reserved words cannot in fact be redefined.    00007000
  The compiler deviates if the program compiles and prints DEVIATES}   00007100
                                                                       00007200
program t6p1p2d2(output);                                              00007300
var                                                                    00007400
   thing:(var,string);                                                 00007500
begin                                                                  00007600
   thing:=string;                                                      00007700
   writeln(' DEVIATES...6.1.2-2, RESERVED WORDS')                      00007800
end.                                                                   00007900


{TEST 6.1.2-3, CLASS=CONFORMANCE}                                      00008000
                                                                       00008100
{ This test checks the implementation of identifiers                   00008200
  and reserved words to see that the two are correctly distinguished.  00008300
  The compiler fails if the program does not compile and               00008400
  print PASS. }                                                        00008500
                                                                       00008600
program t6p1p2d3(output);                                              00008700
var                                                                    00008800
   procedurex,procedurf,procedur:char;                                 00008900
   functionx,functiom,functio:integer;                                 00009000
   iffy:boolean;                                                       00009100
begin                                                                  00009200
   procedurex:='0';                                                    00009300
   procedurf:='1';                                                     00009400
   procedur:='2';                                                      00009500
   functionx:=0;                                                       00009600
   functiom:=1;                                                        00009700
   functio:=2;                                                         00009800
   iffy:=true;                                                         00009900
   writeln(' PASS...6.1.2-3, IDENTIFIERS')                             00010000
end.                                                                   00010100
```

```
{TEST 6.1.3-1, CLASS=CONFORMANCE}                                        00010200
                                                                         00010300
{ The Pascal Standard permits identifiers to be of any length            00010400
  This test will simply print out 'PASS' if the compiler accepts         00010500
  identifiers of lengths up to 70 characters. }                          00010600
                                                                         00010700
program t6p1p3d1(output);                                                00010800
const                                                                    00010900
   i10iiiiiii = 10;                                                      00011000
   i20iiiiiiiiiiiiiiii = 20;                                             00011100
   i30iiiiiiiiiiiiiiiiiiiiiiiiii = 30;                                   00011200
   i40iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii = 40;                        00011300
   i50iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii = 50;                00011400
   i60iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii = 60;        00011500
i70iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii         00011600
                                                              = 70;      00011700
                                                                         00011800
begin                                                                    00011900
   if i10iiiiiii + i20iiiiiiiiiiiiiiii +                                 00012000
      i30iiiiiiiiiiiiiiiiiiiiiiiii +                                     00012100
      i40iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii +                          00012200
      i50iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii +                  00012300
      i60iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii +          00012400
i70iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii         00012500
      <> 280 then                                                        00012600
      writeln(' FAIL...6.1.3-1')                                         00012700
   else                                                                  00012800
      writeln(' PASS...6.1.3-1')                                         00012900
end.                                                                     00013000


{TEST 6.1.3-2, CLASS=CONFORMANCE}                                        00013100
                                                                         00013200
{ The Pascal Standard states that matching upper and lower               00013300
  case letters are equivalent in identifiers and word-symbols            00013400
  (i.e. reserved words) if they are permitted. If this is the            00013500
  case for this compiler, then the program shall print 'PASS'.           00013600
  The compiler fails  if the program cannot be compiled.                 00013700
  This test is irrelevant for one case compilers. }                      00013800
                                                                         00013900
program t6p1p3d2(output);                                                00014000
var                                                                      00014100
   conform  : integer;                                                   00014200
                                                                         00014300
Begin                                                                    00014400
   BEGIN                                                                 00014500
      Conform:=1;                                                        00014600
      CONFORM:=2;                                                        00014700
      If conform = 2 then                                                00014800
         writeln(' PASS...6.1.3-2')                                      00014900
   enD                                                                   00015000
end.                                                                     00015100
```

```
{TEST 6.1.3-3, CLASS=QUALITY}                                            00015200
                                                                         00015300
{ Although the Standard places no limit on the LENGTH of                 00015400
  identifiers, they must be UNIQUE in at least the first 8               00015500
  characters. This program will determine the significance               00015600
  of identifiers from 3 characters to 20 characters.                     00015700
  This program is of course non-standard, and relies on scope            00015800
  masquerades. It assumes a naive significance limit exists,             00015900
  or that there is none. Some compilers may violate the assumption       00016000
  by hashing an identifier tail or preserving the real length. }         00016100
                                                                         00016200
program t6p1p3d3(output);                                                00016300
const                                                                    00016400
   i3i = 3;                                                              00016500
   i4ii = 1;                                                             00016600
   i5iii = 1;                                                            00016700
   i6iiii = 1;                                                           00016800
   i7iiiii = 1;                                                          00016900
   i8iiiiii = 1;                                                         00017000
   i9iiiiiii = 1;                                                        00017100
   i10iiiiiii = 1;                                                       00017200
   i11iiiiiiii = 1;                                                      00017300
   i12iiiiiiii = 1;                                                      00017400
   i13iiiiiiiii = 1;                                                     00017500
   i14iiiiiiiiii = 1;                                                    00017600
   i15iiiiiiiiii = 1;                                                    00017700
   i16iiiiiiiiiii = 1;                                                   00017800
   i17iiiiiiiiiii = 1;                                                   00017900
   i18iiiiiiiiiiii = 1;                                                  00018000
   i19iiiiiiiiiiii = 1;                                                  00018100
   i20iiiiiiiiiiiii = 1;                                                 00018200
                                                                         00018300
procedure signif;                                                        00018400
const                                                                    00018500
   i3j = 0;                                                              00018600
   i4ij = 0;                                                             00018700
   i5iij = 0;                                                            00018800
   i6iiij = 0;                                                           00018900
   i7iiiij = 0;                                                          00019000
   i8iiiiij = 0;                                                         00019100
   i9iiiiiij = 0;                                                        00019200
   i10iiiiiij = 0;                                                       00019300
   i11iiiiiiij = 0;                                                      00019400
   i12iiiiiiij = 0;                                                      00019500
   i13iiiiiiiij = 0;                                                     00019600
   i14iiiiiiiiij = 0;                                                    00019700
   i15iiiiiiiiij = 0;                                                    00019800
   i16iiiiiiiiiij = 0;                                                   00019900
   i17iiiiiiiiiiij = 0;                                                  00020000
   i18iiiiiiiiiiiij = 0;                                                 00020100
   i19iiiiiiiiiiiiij = 0;                                                00020200
   i20iiiiiiiiiiiiiij = 0;                                               00020300
                                                                         00020400
var                                                                      00020500
   x : integer;                                                          00020600
begin                                                                    00020700
   x:=i3i + i4ii + i5iii + i6iiii + i7iiiii + i8iiiiii +                 00020800
      i9iiiiiii + i10iiiiiii + i11iiiiiiii + i12iiiiiiii +               00020900
      i13iiiiiiiii + i14iiiiiiiiii + i15iiiiiiiiii +                     00021000
      i16iiiiiiiiiii + i17iiiiiiiiiii + i18iiiiiiiiiiii +                00021100
      i19iiiiiiiiiiiii + i20iiiiiiiiiiiii;                               00021200
```

```
   if x = 20 then                                                    00021300
      writeln(' NUMBER OF SIGNIFICANT CHARACTERS >= 20')             00021400
   else                                                              00021500
      writeln(' NUMBER OF SIGNIFICANT CHARACTERS = ', x)             00021600
end;                                                                 00021700
                                                                     00021800
begin                                                                00021900
   signif;                                                           00022000
end.                                                                 00022100
                                                                     00022200

{TEST 6.1.5-1, CLASS=CONFORMANCE}                                    00022300
                                                                     00022400
{ This program tests the conformance of the compiler to              00022500
  the syntax productions for numbers specified by the                00022600
  Pascal Standard.                                                   00022700
  If all productions are permitted the program will                  00022800
  print 'PASS'. The compiler fails if the program will              00022900
  not compile. }                                                     00023000
                                                                     00023100
program t6p1p5d1(output);                                            00023200
                                                                     00023300
const                                                                00023400
   {'all cases are legal productions }                               00023500
   a = 1;                                                            00023600
   b = 12;                                                           00023700
   c = 0123;                                                         00023800
   d = 123.0123;                                                     00023900
   e = 123.0123E+2;                                                  00024000
   f = 123.0123E-2;                                                  00024100
   g = 123.0123E2;                                                   00024200
   h = 123E+2;                                                       00024300
   i = 0123E-2;                                                      00024400
   j = 0123E2;                                                       00024500
                                                                     00024600
begin                                                                00024700
   writeln(' PASS...6.1.5-1')                                        00024800
end.                                                                 00024900

{TEST 6.1.5-2, CLASS=CONFORMANCE}                                    00025000
                                                                     00025100
{ This program simply tests if very long numbers are permitted.      00025200
  The value should be representable despite its length. }            00025300
                                                                     00025400
program t6p1p5d2(output);                                            00025500
const                                                                00025600
   reel = 123.456789012345678901234567890123456789;                 00025700
begin                                                                00025800
   writeln(' PASS...6.1.5-2')                                        00025900
end.                                                                 00026000
```

```
{TEST 6.1.5-3, CLASS=DEVIANCE}                                       00026100
                                                                     00026200
{ The number productions specified in the Pascal Standard            00026300
  clearly state that a decimal point must be preceded by             00026400
  a digit sequence.                                                  00026500
  The compiler deviates if the program compiles, in which case       00026600
  the program will print 'DEVIATES', or if one of the cases is       00026700
  accepted.                                                          00026800
  The compiler conforms if all the cases are rejected. }             00026900
                                                                     00027000
program t6p1p5d3(output);                                            00027100
const                                                                00027200
   r = .123;                                                         00027300
var                                                                  00027400
   i : real;                                                         00027500
begin                                                                00027600
   i:=.123;                                                          00027700
   i:=-.123;                                                         00027800
   writeln(' DEVIATES...6.1.5-3');                                   00027900
end.                                                                 00028000

{TEST 6.1.5-4, CLASS=DEVIANCE}                                       00028100
                                                                     00028200
{ The number productions specified in the Pascal Standard            00028300
  clearly state that a decimal point must be followed by             00028400
  a digit sequence.                                                  00028500
  The compiler deviates if the program compiles, in which case       00028600
  the program will print 'DEVIATES'.                                 00028700
  The compiler conforms if the program fails to compile. }           00028800
                                                                     00028900
program t6p1p5d4(output);                                            00029000
var                                                                  00029100
   i : real;                                                         00029200
                                                                     00029300
begin                                                                00029400
   i:=0123.;                                                         00029500
   writeln(' DEVIATES...6.1.5-4');                                   00029600
end.                                                                 00029700

{TEST 6.1.5-5, CLASS=DEVIANCE}                                       00029800
                                                                     00029900
{ Spaces in numbers are forbidden by the Pascal Standard             00030000
  This includes spaces around '.' and 'E'. The compiler              00030100
  deviates if ONE or MORE of the cases below are accepted.           00030200
  The compiler conforms if ALL cases are rejected. }                 00030300
                                                                     00030400
program t6p1p5d5(output);                                            00030500
const                                                                00030600
   one   = 1 234;                                                    00030700
   two   = 0 .1234;                                                  00030800
   three = 0. 1234;                                                  00030900
   four  = 1234 E2;                                                  00031000
   five  = 1234E 2;                                                  00031100
   six   = 1234E- 2;                                                 00031200
   seven = 1234E+ 2;                                                 00031300
begin                                                                00031400
   writeln(' DEVIATES...6.1.5-5')                                    00031500
end.                                                                 00031600
```

```
{TEST 6.1.5-6, CLASS=DEVIANCE}

{ The Pascal standard allows equivalence of upper and
  lower-case letters in names and reserved words only.
  Will the compiler accept 'e' as equivalent to 'E'?
  It should not. The test is not relevant
  to one-case processors. }

program t6p1p5d6(output);
var
   i : real;
begin
   i:=123e2;
   writeln(' DEVIATES...6.1.5-6')
end.


{TEST 6.1.6-1, CLASS=CONFORMANCE}

{ Labels are permitted in standard Pascal. This program
  simply tests if they are permitted by this compiler.
  The compiler fails if the program will not compile
  (or the message printed out is incorrect). }

program t6p1p6d1(output);
label
   1,2,3,4,5;

begin
      write(' P');
      goto 4;
   1: write('.6');
      goto 5;
   2: write('SS');
      goto 3;
   3: write('..');
      goto 1;
   4: write('A');
      goto 2;
   5: writeln('.1.6-1');
end.


{TEST 6.1.6-2, CLASS=CONFORMANCE}

{ Labels should be distinguished by their apparent integral value
  according to the Pascal Standard.
  This program tests if this is the case for this compiler. If so
  then the program shall print PASS. }

program t6p1p6d2(output);
label
   5,6,7;
begin
      goto 5;
0006: goto 7;
   5: goto 6;
 007: writeln('PASS...6.1.6-2')
end.
```

```
{TEST 6.1.7-1, CLASS=CONFORMANCE}

{ Character strings consisting of a single character
  are the constants of the standard type char. This
  program simply tests that these are permitted by
  the compiler.
  The compiler fails if the program will not compile. }

program t6p1p7d1(output);
const
   one = '1';
   two = '2';
var
   twotoo : char;
begin
   if (one <> two) and (two = '2') then
      begin
      twotoo:='2';
      if twotoo = two then
         writeln(' PASS...6.1.7-1')
      else
         writeln(' FAIL...6.1.7-1')
      end
   else
      writeln(' FAIL...6.1.7-1')
end.


{TEST 6.1.7-2, CLASS=CONFORMANCE}

{ The Pascal standard does not place an upper limit
  on the length of strings. This program tests if strings
  are permitted up to a length of 68 characters. The
  compiler fails if the program will not compile. }

program t6p1p7d2(output);
type
   string1 = packed array[1..68] of char;
   string2 = packed array[1..33] of char;
var
   alpha : string1;
   i     : string2;
begin
   alpha:=
'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOP';
   i:='IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII';
   writeln(' PASS...6.1.7-2')
end.
```

```
{TEST 6.1.7-3, CLASS=CONFORMANCE}                                   00041800
                                                                    00041900
{ The Pascal standard allows quotes to appear as char               00042000
  constants and permits them to appear in strings.                  00042100
  If this is desired, they must be written twice.                   00042200
  This program tests that the compiler will allow this.             00042300
  The compiler fails if the program will not compile. }             00042400
                                                                    00042500
program t6p1p7d3(output);                                           00042600
const                                                               00042700
   quote = '''';                                                    00042800
   strquote = 'CAN''T';                                             00042900
begin                                                               00043000
   writeln(' PASS...6.1.7-3')                                       00043100
end.                                                                00043200


{TEST 6.1.7-4, CLASS=DEVIANCE}                                      00043300
                                                                    00043400
{ This program tests that strings of different lengths are          00043500
  not compatible (i.e. 1..m and 1..n).                              00043600
  The compiler fails if the program compiles. }                     00043700
                                                                    00043800
program t6p1p7d4(output);                                           00043900
const                                                               00044000
   string1 = 'STRING1';                                             00044100
var                                                                 00044200
   string2 : packed array[1..5] of char;                            00044300
begin                                                               00044400
   string2:=string1;                                                00044500
   writeln(' DEVIATES...6.1.7-4')                                   00044600
end.                                                                00044700


{TEST 6.1.7-5, CLASS=DEVIANCE}                                      00044800
                                                                    00044900
{ The Pascal Standard specifically states that character            00045000
  strings are constants of the type                                 00045100
      packed array[1..n] of char                                    00045200
  This program tests that this type is not compatible               00045300
  with unpacked arrays.                                             00045400
  The compiler conforms if the program fails to compile. }          00045500
                                                                    00045600
program t6p1p7d5(output);                                           00045700
var                                                                 00045800
   string1 : packed array[1..4] of char;                            00045900
   string2 : array[1..4] of char;                                   00046000
begin                                                               00046100
   string1:='STR1';                                                 00046200
   string2:='STR2';                                                 00046300
   writeln(' DEVIATES...6.1.7-5')                                   00046400
end.                                                                00046500
```

```
{TEST 6.1.7-6, CLASS=DEVIANCE}                                      00046600
                                                                    00046700
{ Again, a character string is a constant of the type               00046800
      packed array[1..n] of char.                                   00046900
  This program tests that strings are not compatible                00047000
  with bounds other than 1..n.                                      00047100
  The compiler conforms if the program fails to compile. }          00047200
                                                                    00047300
program t6p1p7d6(output);                                           00047400
var                                                                 00047500
   string1 : packed array[1..4] of char;                            00047600
   string2 : packed array[0..3] of char;                            00047700
   string3 : packed array[2..5] of char;                            00047800
begin                                                               00047900
   string1:='STR1';                                                 00048000
   string2:='STR2';                                                 00048100
   string3:='STR3';                                                 00048200
   writeln(' DEVIATES...6.1.7-6')                                   00048300
end.                                                                00048400


{TEST 6.1.7-7, CLASS=DEVIANCE}                                      00048500
                                                                    00048600
{ Again, as character strings are constants of the type             00048700
      packed array[1..n] of char,                                   00048800
  they should not be compatible with packed arrays of               00048900
  subranges of char.                                                00049000
  The compiler conforms if the program will not compile. }          00049100
                                                                    00049200
program t6p1p7d7(output);                                           00049300
type                                                                00049400
   alpha = 'A'..'Z';                                                00049500
var                                                                 00049600
   string1 : packed array[1..4] of char;                            00049700
   string2 : packed array[1..4] of alpha;                           00049800
begin                                                               00049900
   string1:='FOUR';                                                 00050000
   string2:='FOUR';                                                 00050100
   writeln(' DEVIATES...6.1.7-7')                                   00050200
end.                                                                00050300
```

```
{TEST 6.1.7-8, CLASS=DEVIANCE}                                          00050400
                                                                        00050500
{ Similarly to 6.1.7-7, subranges of char should not be                 00050600
  compatible with packed arrays of char. However, if the extension      00050700
  is allowed, it should be correctly handled.                           00050800
  Standards conforming processors will not compile the program,         00050900
  compilers which permit the extension should detect the error          00051000
  at compile-time or run-time, but should not execute without           00051100
  error. }                                                              00051200
                                                                        00051300
program t6p1p7d8(output);                                               00051400
type                                                                    00051500
   digit = '0'..'9';                                                    00051600
var                                                                     00051700
   string1 : packed array[1..4] of char;                                00051800
   string2 : packed array[1..4] of digit;                               00051900
begin                                                                   00052000
   string1:='FOUR';                                                     00052100
   string2:='FOUR';                                                     00052200
   writeln(' DEVIATES...6.1.7-8')                                       00052300
end.                                                                    00052400


{TEST 6.1.7-9, CLASS=DEVIANCE}                                          00052500
                                                                        00052600
{ Some compilers may allow compatibility between strings and char       00052700
  constants. The two types for which they are constants are not         00052800
  compatible.                                                           00052900
  This program tests what the compiler will allow.                      00053000
  If all cases are accepted the program will print DEVIATES.            00053100
  However, if one or more of the cases are accepted, then the           00053200
  compiler deviates for those cases. }                                  00053300
                                                                        00053400
program t6p1p7d9(output);                                               00053500
const                                                                   00053600
   a = 'A';                                                             00053700
var                                                                     00053800
   string1 : packed array[1..4] of char;                                00053900
   string2 : packed array[1..1] of char;                                00054000
   achar   : char;                                                      00054100
begin                                                                   00054200
   string1:=a;          { CASE 1 }                                      00054300
   string1:='A';        { CASE 2 }                                      00054400
   string2:=a;          { CASE 3 }                                      00054500
   string2:='A';        { CASE 4 }                                      00054600
   achar:=string2;      { CASE 5 }                                      00054700
   string1:='A   ';                                                     00054800
   achar:=string1;      { CASE 6 }                                      00054900
   string1:='  A';                                                      00055000
   achar:=string1;      { CASE 7 }                                      00055100
   writeln(' DEVIATES...6.1.7-9')                                       00055200
end.                                                                    00055300
```

```
{TEST 6.1.7-10, CLASS=DEVIANCE}                                         00055400
                                                                        00055500
{ The Pascal Standard states that string types are compatible           00055600
  if they have the same number of components.                           00055700
  Some compilers may allow assignment of one string type to another,    00055800
  padding out with spaces or truncating characters if they are          00055900
  not of the same lengths.                                              00056000
  All the cases below should be strictly rejected.                      00056100
  The compiler deviates if one or more are accepted. }                  00056200
                                                                        00056300
program t6p1p7d10(output);                                              00056400
var                                                                     00056500
   string1 : packed array[1..4] of char;                                00056600
   string2 : packed array[1..6] of char;                                00056700
begin                                                                   00056800
   writeln('DEVIATES...6.1.7-10');                                      00056900
   string1:='AB';       { 1-pad with spaces ? }                         00057000
   writeln('CASE 1 : ', string1);                                       00057100
   string1:='ABCD';                                                     00057200
   string2:=string1;    { 2-what happens here ? }                       00057300
   writeln('CASE 2 : ', string2);                                       00057400
   string1:='ABCDEFG';  { 3-what happens here ? }                       00057500
   writeln('CASE 3 : ', string1)                                        00057600
end.                                                                    00057700


{TEST 6.1.7-11, CLASS=DEVIANCE}                                         00057800
                                                                        00057900
{ The Pascal Standard says that a character string is a sequence of     00058000
  characters enclosed by apostrophes, consequently there is no          00058100
  NULL string. Does the compiler allow this in programs.                00058200
  The compiler conforms if the program does not compile. }              00058300
                                                                        00058400
program t6p1p7d11(output);                                              00058500
begin                                                                   00058600
   writeln('':20);                                                      00058700
   writeln(' DEVIATES...6.1.7-11')                                      00058800
end.                                                                    00058900


{TEST 6.1.8-1, CLASS=CONFORMANCE}                                       00059000
                                                                        00059100
{ The Pascal Standard states that a comment is considered to            00059200
  be a token separator. This program tests if the compiler              00059300
  allows this.                                                          00059400
  The compiler fails if the program cannot be compiled. }               00059500
                                                                        00059600
program{ Is this permitted to be here? }t6p1p8d1(output){ Or here? };   00059700
var                                                                     00059800
   i{ control variable }:{ colon }integer{ type };                      00059900
begin                                                                   00060000
   for{ This is a FOR loop }i{ control variable }:={ assignment }       00060100
      1{ initial value }to{ STEP 1 UNTIL }1{ repetitions }do{ go }      00060200
      writeln{ write statement }(' PASS...6.1.8-1')                     00060300
end.                                                                    00060400
```

```
{TEST 6.1.8-2, CLASS=CONFORMANCE}                                    00060500
                                                                     00060600
{ The Pascal Standard permits an open curly bracket to               00060700
  appear in a comment. This program tests that the                   00060800
  compiler will allow this.                                          00060900
  The compiler fails if the program will not compile. }              00061000
                                                                     00061100
program t6p1p8d2(output);                                            00061200
begin                                                                00061300
   { Is a { permitted in a comment? }                                00061400
   writeln(' PASS...6.1.8-2')                                        00061500
end.                                                                 00061600


{TEST 6.1.8-3, CLASS=CONFORMANCE}                                    00061700
                                                                     00061800
{ This program tests that if the compiler allows both                00061900
  forms of comments, must the delimiters be the same.                00062000
  If only one form of comment is permitted, the test is not relevant. } 00062100
                                                                     00062200
program t6p1p8d3(output);                                            00062300
begin                                                                00062400
   { This is a standard comment }                                    00062500
   (* This is an alternative form *)                                00062600
   { What will happen here? *).....}                                 00062700
   (* Or here? }.....*)                                             00062800
   writeln(' PASS...6.1.8-3')                                        00062900
end.                                                                 00063000


{TEST 6.1.8-4, CLASS=QUALITY}                                        00063100
                                                                     00063200
{ In the case of an unclosed comment, does the compiler help         00063300
  the programmer to detect that this is so ? Hard to trace run-time  00063400
  errors may occur  if a comment accidentally encloses 1 or more     00063500
  statements. }                                                      00063600
                                                                     00063700
program t6p1p8d4(output);                                            00063800
var                                                                  00063900
   i : integer;                                                      00064000
begin                                                                00064100
   i:=10;                                                            00064200
   { Now write out the value of i.                                   00064300
   writeln(' THE VALUE OF I IS:', i);                                00064400
   { The value of i will not be printed because of the unclosed      00064500
     previous comment. }                                             00064600
   i:=0                                                              00064700
end.                                                                 00064800


{TEST 6.1.8-5, CLASS=DEVIANCE}                                       00064900
                                                                     00065000
{ Nested comments are not permitted in Pascal and hence              00065100
  this program should not compile. The compiler deviates if the      00065200
  program compiles and prints DEVIATES. }                            00065300
                                                                     00065400
program t6p1p8d5(output);                                            00065500
begin                                                                00065600
   { writeln(' RAN')                                                 00065700
   { writeln(' RAN1') }                                              00065800
   writeln(' RAN2') }                                                00065900
   writeln(' DEVIATES...6.1.8-5, NESTED COMMENTS')                   00066000
end.                                                                 00066100


{TEST 6.2.1-1, CLASS=CONFORMANCE}                                    00066200
                                                                     00066300
{ This program includes a sample of each declaration                 00066400
  part in its minimal form.  Every possibility is covered elsewhere  00066500
  in the validation suite, but the test is made here. }              00066600
                                                                     00066700
program t6p2p1d1(output);                                            00066800
label                                                                00066900
   1;                                                                00067000
const                                                                00067100
   one = 1;                                                          00067200
type                                                                 00067300
   small = 1..3;                                                     00067400
var                                                                  00067500
   tiny : small;                                                     00067600
procedure p(var x : small);                                          00067700
begin                                                                00067800
   x:=1                                                              00067900
end;                                                                 00068000
begin                                                                00068100
   goto 1;                                                           00068200
1: p(tiny);                                                          00068300
   if (tiny = one) then                                              00068400
      writeln(' PASS...6.2.1-1')                                     00068500
end.                                                                 00068600
```

```
{TEST 6.2.1-2, CLASS=CONFORMANCE}                                    00068700
                                                                     00068800
{ This program checks that multiple repetitions are possible         00068900
  in the declaration parts, and is provided as a check.  Practically 00069000
  all occurrences will re-appear elsewhere in the validation         00069100
  suite. }                                                           00069200
                                                                     00069300
program t6p2pld2(output);                                            00069400
label                                                                00069500
   1,2,3;                                                            00069600
const                                                                00069700
   one=1;                                                            00069800
   two=2;                                                            00069900
   three=3;                                                          00070000
type                                                                 00070100
   small = 1..3;                                                     00070200
   larger = 1..10;                                                   00070300
   biggest = 1..100;                                                 00070400
var                                                                  00070500
   tiny : small;                                                     00070600
   soso : larger;                                                    00070700
   big : biggest;                                                    00070800
procedure p(var x : small);                                          00070900
begin                                                                00071000
   x:=1                                                              00071100
end;                                                                 00071200
procedure q(var y : larger);                                         00071300
begin                                                                00071400
   y:=2                                                              00071500
end;                                                                 00071600
procedure r(var z : biggest);                                        00071700
begin                                                                00071800
   z:=3                                                              00071900
end;                                                                 00072000
begin                                                                00072100
   p(tiny); goto 2;                                                  00072200
1: r(big); goto 3;                                                   00072300
2: q(soso); goto 1;                                                  00072400
3: if (tiny=one) and (soso=two) and (big=three) then                 00072500
      writeln(' PASS...6.2.1-2')                                     00072600
end.                                                                 00072700
```

```
{TEST 6.2.1-3, CLASS=DEVIANCE}                                       00072800
                                                                     00072900
{ Checks to see that labels are not permitted unless                 00073000
  they have been declared in the heading }                           00073100
                                                                     00073200
program t6p2pld3(output);                                            00073300
begin                                                                00073400
   3: writeln(' DEVIATES...6.2.1-3')                                 00073500
end.                                                                 00073600
```

```
{TEST 6.2.1-4, CLASS=DEVIANCE}                                       00073700
                                                                     00073800
{ Checks to see that labels may not be given two sites               00073900
  in the executable part.  Since the label is not used               00074000
  in a goto this program is a stringent test. }                      00074100
                                                                     00074200
program t6p2pld4(output);                                            00074300
label                                                                00074400
   9;                                                                00074500
begin                                                                00074600
   9: write(' DEVIATES');                                            00074700
      if true <> false then                                          00074800
         9: writeln('...6.2.1-4')                                    00074900
end.                                                                 00075000
```

```
{TEST 6.2.1-5, CLASS=DEVIANCE}                                       00075100
                                                                     00075200
{ This program declares a label, but it is not sited                 00075300
  nor referenced.  This is illegal, as each declared                 00075400
  label must appear once (and only once) in the executable           00075500
  part of the program. }                                             00075600
                                                                     00075700
program t6p2pld5(output);                                            00075800
label                                                                00075900
   9;                                                                00076000
begin                                                                00076100
   writeln(' DEVIATES...6.2.1-5')                                    00076200
end.                                                                 00076300
```

```
{TEST 6.2.1-6, CLASS=CONFORMANCE}                                    00076400
                                                                     00076500
{ This is the minimal program. }                                     00076600
                                                                     00076700
program t6p2pld6;                                                    00076800
begin                                                                00076900
end.                                                                 00077000
```

```
{TEST 6.2.1-7, CLASS=ERRORHANDLING}                                          00077100
                                                                             00077200
{ The Pascal Standard states that '..local variables have values             00077300
  which are undefined at the beginning of the statement part..'.             00077400
  The undefined value is dependent on the implementation.                    00077500
  Ideally the program should not run. However, if it does, the               00077600
  program shall print the value of i, whether it be a system                 00077700
  initialized value or rubbish left over from procedure q. }                 00077800
                                                                             00077900
program t6p2p1d7(output);                                                    00078000
                                                                             00078100
procedure q;                                                                 00078200
var                                                                          00078300
   i,j : integer;                                                            00078400
begin                                                                        00078500
   i:=2;                                                                     00078600
   j:=3                                                                      00078700
end;                                                                         00078800
                                                                             00078900
procedure r;                                                                 00079000
var                                                                          00079100
   i : integer;                                                              00079200
begin                                                                        00079300
   writeln('ERROR NOT DETECTED...6.2.1-7: THE VALUE OF I IS ', I)            00079400
end;                                                                         00079500
                                                                             00079600
{ Program body }                                                             00079700
begin                                                                        00079800
   q;                                                                        00079900
   r                                                                         00080000
end.                                                                         00080100
```

```
{TEST 6.2.1-8, CLASS=QUALITY}                                                00080200
                                                                             00080300
{ This test checks that a large number of types may be declared              00080400
  in a program. It is an attempt to discover any small limit imposed         00080500
  on the number of types by a compiler. }                                    00080600
                                                                             00080700
program t6p2p1d8(output);                                                    00080800
type                                                                         00080900
   t1 = 0..1;                                                                00081000
   t2 = 0..2;                                                                00081100
   t3 = 0..3;                                                                00081200
   t4 = 0..4;                                                                00081300
   t5 = 0..5;                                                                00081400
   t6 = 0..6;                                                                00081500
   t7 = 0..7;                                                                00081600
   t8 = 0..8;                                                                00081700
   t9 = 0..9;                                                                00081800
   t10 = 0..10;                                                              00081900
   t11 = 0..11;                                                              00082000
   t12 = 0..12;                                                              00082100
   t13 = 0..13;                                                              00082200
   t14 = 0..14;                                                              00082300
   t15 = 0..15;                                                              00082400
   t16 = 0..16;                                                              00082500
   t17 = 0..17;                                                              00082600
   t18 = 0..18;                                                              00082700
   t19 = 0..19;                                                              00082800
   t20 = 0..20;                                                              00082900
   t21 = 0..21;                                                              00083000
   t22 = 0..22;                                                              00083100
   t23 = 0..23;                                                              00083200
   t24 = 0..24;                                                              00083300
   t25 = 0..25;                                                              00083400
   t26 = 0..26;                                                              00083500
   t27 = 0..27;                                                              00083600
   t28 = 0..28;                                                              00083700
   t29 = 0..29;                                                              00083800
   t30 = 0..30;                                                              00083900
   t31 = 0..31;                                                              00084000
   t32 = 0..32;                                                              00084100
   t33 = 0..33;                                                              00084200
   t34 = 0..34;                                                              00084300
   t35 = 0..35;                                                              00084400
   t36 = 0..36;                                                              00084500
   t37 = 0..37;                                                              00084600
   t38 = 0..38;                                                              00084700
   t39 = 0..39;                                                              00084800
   t40 = 0..40;                                                              00084900
   t41 = 0..41;                                                              00085000
   t42 = 0..42;                                                              00085100
   t43 = 0..43;                                                              00085200
   t44 = 0..44;                                                              00085300
   t45 = 0..45;                                                              00085400
   t46 = 0..46;                                                              00085500
   t47 = 0..47;                                                              00085600
   t48 = 0..48;                                                              00085700
   t49 = 0..49;                                                              00085800
   t50 = 0..50;                                                              00085900
var                                                                          00086000
   v1 : t1;                                                                  00086100
   v2 : t2;                                                                  00086200
```

```
   v3 : t3;                                                    00086300
   v4 : t4;                                                    00086400
   v5 : t5;                                                    00086500
   v6 : t6;                                                    00086600
   v7 : t7;                                                    00086700
   v8 : t8;                                                    00086800
   v9 : t9;                                                    00086900
   v10 : t10;                                                  00087000
   v11 : t11;                                                  00087100
   v12 : t12;                                                  00087200
   v13 : t13;                                                  00087300
   v14 : t14;                                                  00087400
   v15 : t15;                                                  00087500
   v16 : t16;                                                  00087600
   v17 : t17;                                                  00087700
   v18 : t18;                                                  00087800
   v19 : t19;                                                  00087900
   v20 : t20;                                                  00088000
   v21 : t21;                                                  00088100
   v22 : t22;                                                  00088200
   v23 : t23;                                                  00088300
   v24 : t24;                                                  00088400
   v25 : t25;                                                  00088500
   v26 : t26;                                                  00088600
   v27 : t27;                                                  00088700
   v28 : t28;                                                  00088800
   v29 : t29;                                                  00088900
   v30 : t30;                                                  00089000
   v31 : t31;                                                  00089100
   v32 : t32;                                                  00089200
   v33 : t33;                                                  00089300
   v34 : t34;                                                  00089400
   v35 : t35;                                                  00089500
   v36 : t36;                                                  00089600
   v37 : t37;                                                  00089700
   v38 : t38;                                                  00089800
   v39 : t39;                                                  00089900
   v40 : t40;                                                  00090000
   v41 : t41;                                                  00090100
   v42 : t42;                                                  00090200
   v43 : t43;                                                  00090300
   v44 : t44;                                                  00090400
   v45 : t45;                                                  00090500
   v46 : t46;                                                  00090600
   v47 : t47;                                                  00090700
   v48 : t48;                                                  00090800
   v49 : t49;                                                  00090900
   v50 : t50;                                                  00091000
begin                                                          00091100
   writeln(' 50 TYPES COMPILED...6.2.1-8')                     00091200
end.                                                           00091300
```

```
{TEST 6.2.1-9, CLASS=QUALITY}                                  00091400
                                                               00091500
{ This test checks that a large number of labels may be declared 00091600
  in a program. It is an attempt to detect a small compiler limit on 00091700
  the number of labels. }                                      00091800
                                                               00091900
program t6p2p1d9(output);                                      00092000
label                                                          00092100
   1,2,3,4,5,6,7,8,9,10,                                       00092200
   11,12,13,14,15,16,17,18,19,20,                              00092300
   21,22,23,24,25,26,27,28,29,30,                              00092400
   31,32,33,34,35,36,37,38,39,40,                              00092500
   41,42,43,44,45,46,47,48,49,50;                              00092600
begin                                                          00092700
   1: ;                                                        00092800
   2: ;                                                        00092900
   3: ;                                                        00093000
   4: ;                                                        00093100
   5: ;                                                        00093200
   6: ;                                                        00093300
   7: ;                                                        00093400
   8: ;                                                        00093500
   9: ;                                                        00093600
   10: ;                                                       00093700
   11: ;                                                       00093800
   12: ;                                                       00093900
   13: ;                                                       00094000
   14: ;                                                       00094100
   15: ;                                                       00094200
   16: ;                                                       00094300
   17: ;                                                       00094400
   18: ;                                                       00094500
   19: ;                                                       00094600
   20: ;                                                       00094700
   21: ;                                                       00094800
   22: ;                                                       00094900
   23: ;                                                       00095000
   24: ;                                                       00095100
   25: ;                                                       00095200
   26: ;                                                       00095300
   27: ;                                                       00095400
   28: ;                                                       00095500
   29: ;                                                       00095600
   30: ;                                                       00095700
   31: ;                                                       00095800
   32: ;                                                       00095900
   33: ;                                                       00096000
   34: ;                                                       00096100
   35: ;                                                       00096200
   36: ;                                                       00096300
   37: ;                                                       00096400
   38: ;                                                       00096500
   39: ;                                                       00096600
   40: ;                                                       00096700
   41: ;                                                       00096800
   42: ;                                                       00096900
   43: ;                                                       00097000
   44: ;                                                       00097100
   45: ;                                                       00097200
   46: ;                                                       00097300
   47: ;                                                       00097400
```

```
   48: ;                                                        00097500
   49: ;                                                        00097600
   50: ;                                                        00097700
   writeln(' 50 LABELS DECLARED AND SITED...6.2.1-9')           00097800
end.                                                            00097900


{TEST 6.2.2-1, CLASS=CONFORMANCE}                               00098000
                                                                00098100
{ The Pascal Standard permits redefinition of a user name, by a 00098200
  further defining occurrence in a range (eg. procedure block)  00098300
  enclosed by the first defining occurence. This second range   00098400
  (and all ranges enclosed by it) are excluded from the scope of 00098500
  the defining occurence of the first range.                    00098600
  This program tests the scope conformance of the compiler      00098700
  for user names. }                                             00098800
                                                                00098900
program t6p2p2d1(output);                                       00099000
const                                                           00099100
   range = 10;                                                  00099200
var                                                             00099300
   i : integer;                                                 00099400
   pass : boolean;                                              00099500
procedure redefine;                                             00099600
const                                                           00099700
   range = -10;                                                 00099800
var                                                             00099900
   i : integer;                                                 00100000
begin                                                           00100100
   i:=range;                                                    00100200
end;                                                            00100300
                                                                00100400
begin                                                           00100500
   i:=1;                                                        00100600
   pass:=false;                                                 00100700
   redefine;                                                    00100800
   if range=-10 then                                            00100900
      writeln(' FAIL...6.2.2-1: SCOPE ERROR-RANGE')             00101000
   else                                                         00101100
      pass:=true;                                               00101200
   if i=-10 then                                                00101300
      writeln(' FAIL...6.2.2-1: SCOPE ERROR-I')                 00101400
   else                                                         00101500
      if pass then                                              00101600
         writeln(' PASS...6.2.2-1')                             00101700
end.                                                            00101800
```

```
{TEST 6.2.2-2, CLASS=CONFORMANCE}                               00101900
                                                                00102000
{ The Pascal Standard allows a user to redefine a predefined name. 00102100
  This program tests whether this is allowed by this compiler. } 00102200
                                                                00102300
program t6p2p2d2(output);                                       00102400
var                                                             00102500
   true : boolean;                                              00102600
begin                                                           00102700
   true:=false;                                                 00102800
   if true = false then                                         00102900
      writeln(' PASS...6.2.2-2')                                00103000
   else                                                         00103100
      writeln(' FAIL...6.2.2-2')                                00103200
end.                                                            00103300


{TEST 6.2.2-3,CLASS=CONFORMANCE}                                00103400
                                                                00103500
{ This program is similar to 6.2.2-4, however a type identifier, 00103600
  say T, which specifies the domain of a pointer type ↑T, is    00103700
  permitted to have its defining occurence anywhere in the type 00103800
  definition part in which ↑T occurs.                           00103900
  Thus in this example, (node=real)s' scope is excluded from the 00104000
  type definition of ouch.                                      00104100
  The compiler fails if the program does not compile or fails at 00104200
  run time. }                                                   00104300
                                                                00104400
program t6p2p2d3(output);                                       00104500
type                                                            00104600
   node = real;                                                 00104700
procedure ouch;                                                 00104800
type                                                            00104900
   p = ↑node;                                                   00105000
   node = boolean;                                              00105100
var                                                             00105200
   ptr : p;                                                     00105300
begin                                                           00105400
   new(ptr);                                                    00105500
   ptr↑:=true;                                                  00105600
   writeln(' PASS...6.2.2-3')                                   00105700
end;                                                            00105800
begin                                                           00105900
   ouch;                                                        00106000
end.                                                            00106100
```

```
{TEST 6.2.2-4, CLASS=DEVIANCE}

{ The Pascal Standard says that the defining occurrence of an
  identifier or label precedes all corresponding occurrences of
  that identifier or label in the program text (except for specific
  pointercase). The scope of an identifier or label also includes
  the whole block in which it is defined, thereby disallowing
  any references to an outer identifier of the same name preceeding
  the defining occurrence.  Some compilers may not conform
  to this and allow some scope overlap.
  The compiler conforms if the program does not compile and objects
  to the use of 'red' in ouch preceding its definition. }

program t6p2p2d4(output);
const
   red = 1;
   violet = 2;
procedure ouch;
const
   m = red;
   n = violet;
type
   a = array[m..n] of integer;
var
   v : a;
   colour : (yellow,green,blue,red,indigo,violet);
begin
   v[1]:=1;
   colour:=red;
end;
begin
   ouch;
   writeln(' DEVIATES...6.2.2-4 --> SCOPE ERROR NOT DETECTED')
end.
```

```
{TEST 6.2.2-5, CLASS=CONFORMANCE}

{ Similarly to 6.2.2-2, labels are allowed to be redefined
  in a range enclosed by the first defining occurrence
  (eg. procedures and functions). This program tests if
  this is permitted by this compiler. }

program t6p2p2d5(output);
label
   4,5,6;
var
   i : integer;
procedure redefine;
label
   6,7,8;
var
   j : integer;
begin
      j:=1;
      goto 6;
   7: j:=j-1;
      goto 8;
   6: j:=j+1;
      goto 7;
   8: j:=0;
end;

begin
      goto 4;
   5: i:=i+1;
      goto 6;
   4: i:=1;
      redefine;
      goto 5;
   6: if i=1 then
         writeln(' FAIL...6.2.2-5')
      else
         writeln(' PASS...6.2.2-5')
end.
```

```
{TEST 6.2.2-6, CLASS=CONFORMANCE}                                    00113500
                                                                     00113600
{ As for the other conformance tests in this section,                00113700
  it is possible to redefine a field-name of a record within         00113800
  the same scope as this record.                                     00113900
  The compiler also fails if the program does not compile. }         00114000
                                                                     00114100
program t6p2p2d6(output);                                            00114200
var                                                                  00114300
   j : integer;                                                      00114400
   x : record                                                        00114500
         j:integer                                                   00114600
       end;                                                          00114700
begin                                                                00114800
   j:=1;                                                             00114900
   x.j:=2;                                                           00115000
   with x do                                                         00115100
      j:=3;                                                          00115200
   if (j=1) and (x.j=3) then writeln(' PASS...6.2.2-6')              00115300
   else  writeln(' FAIL...6.2.2-6')                                  00115400
end.                                                                 00115500


{TEST 6.2.2-7, CLASS=DEVIANCE}                                       00115600
                                                                     00115700
{ It is possible to redefine a function name within the scope        00115800
  of a function name. This test checks that the inner function       00115900
  redefines f, whether an erroneous assignment to f is detected or   00116000
  whether the erroneous outer f, with no function assignment is      00116100
  allowed to execute. }                                              00116200
                                                                     00116300
program t6p2p2d7(output);                                            00116400
var                                                                  00116500
   bool:boolean;                                                     00116600
   j:integer;                                                        00116700
                                                                     00116800
function f(i:integer) : integer;                                     00116900
   function f(i:integer) : integer;                                  00117000
   begin                                                             00117100
      f:=i                                                           00117200
   end;                                                              00117300
begin                                                                00117400
   if bool then                                                      00117500
      writeln(' FAIL...6.2.2-7, PROCEDURE SCOPE')                    00117600
      { FAILs if the call is recursive }                             00117700
   else begin                                                        00117800
      bool:=true;                                                    00117900
      f:=f(i);                                                       00118000
   end                                                               00118100
end;                                                                 00118200
                                                                     00118300
begin                                                                00118400
   bool:=false;                                                      00118500
   j:=f(1);                                                          00118600
   if (j=1) then                                                     00118700
      writeln(' DEVIATES...6.2.2-7, PROCEDURE SCOPE');               00118800
end.                                                                 00118900
```

```
{TEST 6.2.2-8, CLASS=CONFORMANCE}                                    00119000
                                                                     00119100
{ It is possible to declare a function but not assign a value        00119200
  to that function at that level. This program assigns a value       00119300
  to a function from within a function within the function.          00119400
  The compiler fails if the program does not compile or it prints    00119500
  FAIL. }                                                            00119600
                                                                     00119700
program t6p2p2d8(output);                                            00119800
var                                                                  00119900
   j,k:integer;                                                      00120000
                                                                     00120100
function f1(i:integer):integer;                                      00120200
   function f2(i:integer):integer;                                   00120300
      function f3(i:integer):integer;                                00120400
      begin                                                          00120500
         f3:=1;                                                      00120600
         f1:=i                                                       00120700
      end;                                                           00120800
   begin                                                             00120900
      f2:=f3(i)                                                      00121000
   end;                                                              00121100
begin                                                                00121200
   j:=f2(i)                                                          00121300
end;                                                                 00121400
                                                                     00121500
begin                                                                00121600
   k:=f1(5);                                                         00121700
   if (k=5) then                                                     00121800
      writeln(' PASS...6.2.2-8, FUNCTION')                           00121900
   else                                                              00122000
      writeln(' FAIL...6.2.2-8, FUNCTION')                           00122100
end.                                                                 00122200
```

```
{TEST 6.2.2-9, CLASS=DEVIANCE}

{ This program attempts to assign a value to a function outside
  the bounds of the function. The compiler deviates if the
  program prints DEVIATES. }

program t6p2p2d9(output);
var
   i:integer;

function f1:integer;
begin
   f1:=6
end;

function f2(i:integer):integer;
begin
   f2:=i;
   f1:=5
end;

begin
   i:=f1;
   i:=f2(2);
   writeln(' DEVIATES...6.2.2-9, FUNCTION')
end.
```

```
{TEST 6.2.2-10, CLASS=CONFORMANCE}

{ This obscure program is nevertheless standard Pascal.
  An inner scope hides part of a type while leaving other parts
  accessible. The compiler fails if the program does not
  compile or the program prints FAIL. }

program t6p2p2d10(output);
type
   colour=(red,amber,green);
var
   c:colour;

procedure nested;
type
   colour=(purple,red,blue);
var
   paint:colour;
begin
   c:=green;
   paint:=red;
   c:=pred(amber);
   if (ord(c)<>0) or (ord(paint)<>1) then
      writeln(' FAIL...6.2.2-10, SCOPE');
end;

begin
   nested;
   if (c<> red) then
      writeln(' FAIL...6.2.2-10, SCOPE')
   else
      writeln(' PASS...6.2.2-10, SCOPE')
end.
```

```
{TEST 6.3-1, CLASS=CONFORMANCE}

{ This program exhibits all legal productions for a constant
  in a const declaration. }

program t6p3d1(output);
const
   ten = 10;
   minusten = -10;
   minustentoo = -ten;
   decade = ten;
   dot = '.';
   stars = '****';
   on = true;
   pi = 3.1415926;
   minuspi = - pi;
begin
   writeln(' PASS...6.3-1')
end.
```

```
{TEST 6.3-2, CLASS=DEVIANCE}

{ This program checks that signed chars are not permitted.
  Note that minus may have a worse effect than plus. }

program t6p3d2(output);
const
   dot = '.';
   plusdot = + dot;
begin
   writeln(' DEVIATES...6.3-2')
end.


{TEST 6.3-3, CLASS=DEVIANCE}

{ This program checks that signed strings are not permitted.
  Note that minus may have a worse effect than plus. }

program t6p3d3(output);
const
   stars = '****';
   plusstars = + stars;
begin
   writeln(' DEVIATES...6.3-3')
end.


{TEST 6.3-4, CLASS=DEVIANCE}

{ This program checks that signed scalars are not permitted.
  Note than minus may have a worse effect than plus. }

program t6p3d4(output);
const
   truth = true;
   plustruth = + truth;
begin
   writeln(' DEVIATES...6.3-4')
end.


{TEST 6.3-5, CLASS=DEVIANCE}

{ This program tests that signed constants are not permitted
  in other contexts than const declarations. }

program t6p3d5(output);
const
   dot = '.';
begin
   writeln(' DEVIATES', +dot, '..6.3-5')
end.
```

```
{TEST 6.3-6, CLASS=DEVIANCE}

{ A constant may not be used in its own declaration - the
  following is a pathological case which should be detected
  or at least handled with care. }

program t6p3d6(output);
const
   ten = 10;

procedure p;
const
   ten = ten;
begin
   if ten=10 then
      writeln(' DEVIATES...6.3-6: SCOPE ERROR')
   else
      writeln(' DEVIATES...6.3-6: DEFINITION POINT ERROR')
end;

begin
   p
end.


{TEST 6.4.1-1, CLASS=CONFORMANCE}

{ This program tests to see that pointer types can be
  declared anywhere in the type part.  This freedom
  is explicitly permitted in the standard. }

program t6p4pld1(output);
type
   ptr1     = ↑ polar;
   polar    = record r,theta : real end;
   purelink = ↑ purelink;
   ptr2     = ↑ person;
   ptr3     = ptr2;
   person   = record
                 mother,father : ptr2;
                 firstchild    : ptr2;
                 nextsibling   : ptr3
              end;
begin
   writeln(' PASS...6.4.1-1')
end.
```

```
{TEST 6.4.1-2, CLASS=DEVIANCE}

{ This program tests that attempts to use types in their
  own definitions are detected.  Two examples are
  attempted. Both should fail. }

program t6p4p1d2(output);
type
   x  = record
           xx : x
        end;
   y  = array[0..1] of y;
begin
   writeln(' DEVIATES...6.4.1-2')
end.


{TEST 6.4.1-3, CLASS=DEVIANCE}

{ This program also tests that attempts to use types in
  their own definitions are detected, but inserts a nasty
  scope twist by making another type with the same identifier
  available in an outer scope.  It should be excluded from this
  scope, according to the Standard. }

program t6p4p1d3(output);
type
   x  = integer;

procedure p;
type
   x  = record
           y : x
        end;
begin
   writeln(' DEVIATES...6.4.1-3: SCOPE ERROR')
end;

begin
   p
end.
```

```
{TEST 6.4.2.2-1, CLASS=CONFORMANCE}

{ This program tests that the standard simple types have all
  been implemented. They are denoted by predefined type identifiers.
  The compiler fails if the program does not compile. }

program t6p4p2p2d1(output);
var
   a : integer;
   b : real;
   c : boolean;
   d : char;
begin
   a:=6*2+3;
   b:=3.14159*2;
   c:=(a=15);
   d:='Z';
   writeln(' PASS...6.4.2.2-1')
end.


{TEST 6.4.2.2-2, CLASS=CONFORMANCE}

{ The Pascal Standard specifies that the values an integer may
  take are within the range -maxint..+maxint.
  This program checks this. }

program t6p4p2p2d2(output);
type
   natural = 0..maxint;
   whole = -maxint..+maxint;
var
   i : natural;
   j : whole;
   k : integer;
begin
   i:=maxint;
   j:=-maxint;
   k:=maxint;
   writeln(' PASS...6.4.2.2-2')
end.


{TEST 6.4.2.2-3, CLASS=CONFORMANCE}

{ The Pascal Standard states that type BOOLEAN has truth values
  denoted by the identifiers true and false, and that they are
  such that false is less than true.
  This program tests if the compiler allows this. }

program t6p4p2p2d3(output);
begin
   if false < true then
      writeln(' PASS...6.4.2.2-3')
   else
      writeln(' FAIL...6.4.2.2-3')
end.
```

```
{TEST 6.4.2.2-4, CLASS=CONFORMANCE}

{ The Pascal Standard states that the character values representing
  the digits 0..9 are ordered and contiguous.
  The program tests these two criteria for these characters. }

program t6p4p2p2d4(output);
var
   a,b : boolean;
begin
   a:=(succ('0') = '1') and
      (succ('1') = '2') and
      (succ('2') = '3') and
      (succ('3') = '4') and
      (succ('4') = '5') and
      (succ('5') = '6') and
      (succ('6') = '7') and
      (succ('7') = '8') and
      (succ('8') = '9') ;

   b:=('0' < '1') and
      ('1' < '2') and
      ('2' < '3') and
      ('3' < '4') and
      ('4' < '5') and
      ('5' < '6') and
      ('6' < '7') and
      ('7' < '8') and
      ('8' < '9') ;
   if a and b then
      writeln(' PASS...6.4.2.2-4')
   else
      writeln(' FAIL...6.4.2.2-4')
end.


{TEST 6.4.2.2-5, CLASS=CONFORMANCE}

{ The Pascal Standard states that the upper-case letters A-Z are
  ordered, but not necessarily contiguous.
  This program determines if this is so, and prints
  a message as to whether the compiler passes or not . }

program t6p4p2p2d5(output);
begin
   if ('A' < 'B') and ('B' < 'C') and ('C' < 'D') and
      ('D' < 'E') and ('E' < 'F') and ('F' < 'G') and
      ('G' < 'H') and ('H' < 'I') and ('I' < 'J') and
      ('J' < 'K') and ('K' < 'L') and ('L' < 'M') and
      ('M' < 'N') and ('N' < 'O') and ('O' < 'P') and
      ('P' < 'Q') and ('Q' < 'R') and ('R' < 'S') and
      ('S' < 'T') and ('T' < 'U') and ('U' < 'V') and
      ('V' < 'W') and ('W' < 'X') and ('X' < 'Y') and
      ('Y' < 'Z') then
      writeln(' PASS...6.4.2.2-5')
   else
      writeln(' FAIL...6.4.2.2-5: NO ORDERING')
end.
```

```
{TEST 6.4.2.2-6, CLASS=CONFORMANCE}

{ The Pascal Standard states that the lower-case letters a-z are
  ordered, but not necessarily contiguous.
  This program determines if this is so, and prints
  a message as to whether the compiler passes or not .
   NOTE: this program uses lower-case char constants and may
         fail for this reason. The test is also irrelevant for
         one-case compilers. }

program t6p4p2p2d6(output);
begin
   if ('a' < 'b') and ('b' < 'c') and ('c' < 'd') and
      ('d' < 'e') and ('e' < 'f') and ('f' < 'g') and
      ('g' < 'h') and ('h' < 'i') and ('i' < 'j') and
      ('j' < 'k') and ('k' < 'l') and ('l' < 'm') and
      ('m' < 'n') and ('n' < 'o') and ('o' < 'p') and
      ('p' < 'q') and ('q' < 'r') and ('r' < 's') and
      ('s' < 't') and ('t' < 'u') and ('u' < 'v') and
      ('v' < 'w') and ('w' < 'x') and ('x' < 'y') and
      ('y' < 'z') then
      writeln(' PASS...6.4.2.2-6')
   else
      writeln(' FAIL...6.4.2.2-6: NO ORDERING')
end.


{TEST 6.4.2.2-7, CLASS=IMPLEMENTATIONDEFINED}

{ The Pascal Standard states that the value of maxint is
  dependent on the implementation.
  This program prints out the implementation defined value
  of maxint. }

program t6p4p2p2d7(output);
begin
   writeln(' THE IMPLEMENTATION DEFINED VALUE OF MAXINT IS
           maxint)
end.


{TEST 6.4.2.3-1, CLASS=CONFORMANCE}

{ This program checks the possible syntax productions for
  enumerated types, as specified by the Pascal Standard.
  The compiler fails if the program does not compile. }

program t6p4p2p3d1(output);
type
   singularitytype = (me);
   switch          = (on,off);
   maritalstatus   = (married,divorced,widowed,single);
   colour          = (red,pink,orange,yellow,green);
   cardsuit        = (heart,diamond,spade,club);
var
   i : singularitytype;
begin
   i:=me;
   writeln(' PASS...6.4.2.3-1')
end.
```

```
{TEST 6.4.2.3-2, CLASS=CONFORMANCE}

{ The Pascal Standard states that the ordering of the values
  of the enumerated type is determined by the sequence in which
  the constants are listed, the first being before the last.
  The compiler fails if the program does not compile. }

program t6p4p2p3d2(output);
var
   suit : (club,spade,diamond,heart);
   a    : boolean;
   b    : boolean;
begin
   a:=(succ(club)=spade) and
      (succ(spade)=diamond) and
      (succ(diamond)=heart);

   b:=(club < spade) and
      (spade < diamond) and
      (diamond < heart);

   if a and b then
      writeln(' PASS...6.4.2.3-2')
   else
      writeln(' FAIL...6.4.2.3-2')
end.


{TEST 6.4.2.4-1, CLASS=CONFORMANCE}

{ This program tests that a type may be defined as a subrange
  of another ordinal-type (host-type).
  The compiler fails if one or more of the cases below are rejected. }

program t6p4p2p4d1(output);
type
   colour      = (red,pink,orange,yellow,green,blue);
   somecolour  = red..green;
   century     = 1..100;
   twentyone   = -10..+10;
   digits      = '0'..'9';
   zero        = 0..0;
   logical     = false..true;
var
   tf : logical;

begin
   tf:=true;
   writeln(' PASS...6.4.2.4-1')
end.
```

```
{TEST 6.4.2.4-2, CLASS=DEVIANCE}

{ This program tests to see if real constants are permitted
  in a subrange declaration. The Pascal Standard states that
  a subrange definition must be of a subrange of another ordinal
  type. This rules out real constants in the definition. }

program t6p4p2p4d2(output);
type
   wiregauge = 0.001..0.2;
begin
   writeln(' DEVIATES...6.4.2.4-2')
end.


{TEST 6.4.2.4-3, CLASS=DEVIANCE}

{ The Pascal Standard states that the first constant in a definition
  specifies the lower bound, which is less than or equal to the
  upper bound.
  This program tests the compilers' conformance to this point.
  The compiler conforms if both the cases are rejected. }

program t6p4p2p4d3(output);
type
   mixedup = 100..0;
   reverse = 'Z'..'A';
begin
   writeln(' DEVIATES...6.4.2.4-3 : EMPTY SUBRANGES ALLOWED')
end.


{TEST 6.4.3.1-1, CLASS=DEVIANCE}

{ The Pascal Standard states that only structured types may be
  PACKED (array, set, file and record types).
  This program tests this point. The compiler conforms if
  the program will not compile. }

program t6p4p3p1d1(output);
type
   switch = packed(on,off);
   state  = packed(high,low,invalid);
   decade   = packed 0..10;
begin
   writeln(' DEVIATES...6.4.3.1-1 : IMPROPER USE OF PACKED')
end.
```

```
{TEST 6.4.3.1-2, CLASS=DEVIANCE}

{ The Pascal Standard states that a structured type identifier
  may not be used in a PACKED type definition.
  The compiler passes if the program fails to compile. }

program t6p4p3pld2(output);
type
   complex = record
               realpart : real;
               imagpart : real;
             end;
   packcom = packed complex;
begin
   writeln(' DEVIATES...6.4.3.1-2 : IMPROPER USE OF PACKED')
end.


{TEST 6.4.3.1-3, CLASS=CONFORMANCE}

{ The Pascal Standard allows array, set, file and
  record types to be declared as PACKED.
  The program simply tests that all these are
  permitted.
  The compiler fails if the program will not compile. }

program t6p4p3pld3(output);
type
   urray    = packed array[1..10] of char;
   rekord   = packed record
                   bookcode : integer;
                   authorcode : integer;
              end;
   fyle     = packed file of urray;
   card     = (heart,diamond,spade,club);
   sett     = packed set of card;
begin
   writeln(' PASS...6.4.3.1-3')
end.


{TEST 6.4.3.2-1, CLASS=CONFORMANCE}

{ This program tests all the valid productions for an
  array declaration from the syntax specified by the
  Pascal Standard.
  The compiler fails if one or more cases are rejected. }

program t6p4p3p2dl(output);
type
   cards       = (two,three,four,five,six,seven,eight,nine,ten,jack,
                  queen,king,ace);
   suit        = (heart,diamond,spade,club);
   hand        = array[cards] of suit;
   picturecards= array[jack..king] of suit;
   played      = array[cards] of array[heart..diamond] of boolean;
   playedtoo   = array[cards,heart..diamond] of boolean;
begin
   writeln(' PASS...6.4.3.2-1')
end.
```

```
{TEST 6.4.3.2-2, CLASS=DEVIANCE}

{ The Pascal Standard states that an index-type must be an
  ordinal-type. This does not include REAL.
  This program tests if the compiler will allow real bounds. }

program t6p4p3p2d2(output);
type
   reeltest = array[1.5..10.1] of real;
begin
   writeln(' DEVIATES...6.4.3.2-2')
end.


{TEST 6.4.3.2-3, CLASS=CONFORMANCE}

{ An index type may be an ordinal type, This allows
  the use of BOOLEAN, INTEGER and some userdefined type
  names to be used as an index type.
  This program tests if the compiler will permit these
  except for INTEGER, which is included in a separate program. }

program t6p4p3p2d3(output);
type
   digits   = '0'..'9';
   colour   = (red,pink,orange,yellow);
   intensity  = (bright,dull);
var
   alltoo   : array[boolean] of boolean;
   numeric  : array[digits] of integer;
   colours  : array[colour] of intensity;
   code     : array[char] of digits;
begin
   numeric['0']:=0;
   colours[pink]:=bright;
   alltoo[true]:=false;
   code['A']:='0';
   writeln(' PASS...6.4.3.2-3')
end.
```

```
{TEST 6.4.3.2-4, CLASS=QUALITY}

{ As mentioned in 6.4.3.2-3, an index type is an ordinal type,
  thus INTEGER may appear as an index type. However on most
  machines this would represent an unusually large array, and
  thus may not be allowed by the compiler.
  This program tests if such a declaration is permitted, and if
  not, is the diagnostic appropriate. }

program t6p4p3p2d4(output);
type
   everything = array[integer] of integer;
var
   all   : everything;
begin
   all[maxint]:=1;
   all[0]:=1;
   all[-maxint]:=1;
   writeln(' QUALITY...6.4.3.2-4: -->INTEGER BOUNDS PERMITTED')
end.


{TEST 6.4.3.2-5, CLASS=DEVIANCE}

{ Strings must have a subrange of integers as an index type.
  The compiler deviates if this program compiles and
  prints DEVIATES. }

program t6p4p3p2d5(output);
type
   colour = (red,blue,yellow,green);
   c11 = blue..green;
var
   s:packed array[c11] of char;
begin
   s:='ABC';
   writeln(' DEVIATES...6.4.3.2-5, INDEX TYPE')
end.
```

```
{TEST 6.4.3.3-1, CLASS=CONFORMANCE}

{ This program simply tests that all valid productions from
  the syntax for record types (as specified by the Pascal Standard)
  are accepted by this compiler.
  The compiler fails if one or more cases are rejected. }

program t6p4p3p3d1(output);
type
   string   = packed array[1..25] of char;
   married  = (false,true);
   shape    = (triangle,rectangle,square,circle);
   angle    = 0..90;
   a        = record
                year : integer;
                month : 1..12;
                day : 1..31
              end;
   b        = record
                name,firstname : string;
                age : 0..99;
                case  married of
                   true: (spousename : string);
                   false : ()
              end;
   c        = record
                case s : shape of
                   triangle : (side : real;
                               inclination,angle1,angle2 : angle);
                   square,rectangle : (side1,side2 : real;
                                       skew,angle3 : angle);
                   circle : (diameter : real)
              end;
   d        = record ; end;
   e        = record
                case married of
                   true : (spousename : string);
                   false : ();
                end;
begin
   writeln(' PASS...6.4.3.3-1')
end.
```

```
{TEST 6.4.3.3-2, CLASS=CONFORMANCE}

{ The Pascal Standard states that the occurrence of a field
  identifier within the identifier list of a record section is
  its defining occurence as a field identifier for the record
  type in which the record section occurs.
  This should allow redefinition of a field identifier in another
  type declaration.
  The compiler fails if the program does not compile. }

program t6p4p3p3d2(output);
type
   a     = record
             realpart : real;
             imagpart : real
           end;
   realpart = (notimaginary,withbody,withsubstance);
begin
   writeln(' PASS...6.4.3.3-2')
end.


{TEST 6.4.3.3-3, CLASS=CONFORMANCE}

{ The Pascal Standard permits the declaration of an empty record,
  this empty record serves little purpose, and for this reason
  some compilers will not allow it to be used.
  The compiler fails if the program does not compile. }

program t6p4p3p3d3(output);
type
   statuskind  = (defined,undefined);
   emptykind   = record end;
var
   empty : emptykind;
   number: record
            case status:statuskind of
               defined  : (i : integer);
               undefined: (e : emptykind)
            end;
begin
   with number do
   begin
      status:=defined;
      i:=7
   end;
   writeln(' PASS...6.4.3.3-3')
end.
```

```
{TEST 6.4.3.3-4, CLASS=CONFORMANCE}

{ Similarly to 6.4.3.3-2, a tag-field may be redefined
  elsewhere in the declaration part.
  The compiler fails if the program will not compile. }

program t6p4p3p3d4(output);
type
   which = (white,black,warlock,sand);
var
   polex : record
             case which:boolean of
               true: (realpart:real;
                      imagpart:real);
               false:(theta:real;
                      magnit:real)
           end;
begin
   polex.which:=true;
   polex.realpart:=0.5;
   polex.imagpart:=0.8;
   writeln(' PASS...6.4.3.3-4')
end.


{TEST 6.4.3.3-5, CLASS=ERRORHANDLING}

{ The Pascal Standard states that if a change of variant occurs
  (by assigning a value associated with a  variant to the
  tag-field), then the fields associated with the previous variants
  cease to exist. This program causes the error to occur. }

program t6p4p3p3d5(output);
type
   two = (a,b);
var
   variant : record
               case tagfield:two of
                  a: (m:integer);
                  b: (n:integer);
             end;
   i : integer;
begin
   variant.tagfield:=a;
   variant.m:=1;
   i:=variant.n;     {illegal}
   writeln(' ERROR NOT DETECTED...6.4.3.3-5')
end.
```

```
{TEST 6.4.3.3-6, CLASS=ERRORHANDLING}                                   00195000
                                                                        00195100
{ The program causes an error by accessing a field with                 00195200
  an undefined value. The undefinition arises because                   00195300
  when a change of variant occurs, those fields associated with         00195400
  the new variant come into existence with undefined                    00195500
  values. }                                                             00195600
                                                                        00195700
program t6p4p3p3d6(output);                                             00195800
type                                                                    00195900
   two = (a,b);                                                         00196000
var                                                                     00196100
   variant : record                                                     00196200
               case tagfield:two of                                     00196300
                  a : (m : integer;                                     00196400
                       l : integer);                                    00196500
                  b : (n : integer;                                     00196600
                       o : integer)                                     00196700
             end;                                                       00196800
   i : integer;                                                         00196900
begin                                                                   00197000
   variant.tagfield:=a;                                                 00197100
   variant.m:=1;                                                        00197200
   variant.l:=1;                                                        00197300
   variant.tagfield:=b;                                                 00197400
   variant.n:=1;                                                        00197500
   i:=variant.o;     { illegal }                                        00197600
   writeln(' ERROR NOT DETECTED...6.4.3.3-6')                           00197700
end.                                                                    00197800


{TEST 6.4.3.3-7, CLASS=ERRORHANDLING}                                   00197900
                                                                        00198000
{ This test is similar to 6.4.3.3-5, except that no tagfield is         00198100
  used.                                                                 00198200
  Variant changes occur implicitly as a result of                       00198300
  assignment to fields. The fields associated with the new              00198400
  variant come into existence with undefined values. }                  00198500
                                                                        00198600
program t6p4p3p3d7(output);                                             00198700
type                                                                    00198800
   two = (a,b);                                                         00198900
var                                                                     00199000
   variant : record                                                     00199100
               case two of                                              00199200
                  a : (m : integer);                                    00199300
                  b : (n : integer);                                    00199400
             end;                                                       00199500
   i : integer;                                                         00199600
begin                                                                   00199700
   variant.m:=2;                                                        00199800
   i:=variant.n;     { illegal }                                        00199900
   writeln(' ERROR NOT DETECTED...6.4.3.3-7')                           00200000
end.                                                                    00200100
```

```
{TEST 6.4.3.3-8, CLASS=ERRORHANDLING}                                   00200200
                                                                        00200300
{ Similar to 6.4.3.3-5, except that no tag-field is used.               00200400
  A change of variant occurs by reference to a field associated         00200500
  with a new variant. Again, these fields come into existance           00200600
  undefined.                                                            00200700
  The compiler conforms if the program does not compile. }              00200800
                                                                        00200900
program t6p4p3p3d8(output);                                             00201000
type                                                                    00201100
   two = (a,b);                                                         00201200
var                                                                     00201300
   variant : record                                                     00201400
               case two of                                              00201500
                  a:(m:integer;                                         00201600
                     l:integer);                                        00201700
                  b:(n:integer;                                         00201800
                     o:integer)                                         00201900
             end;                                                       00202000
   i : integer;                                                         00202100
begin                                                                   00202200
   variant.n:=1;                                                        00202300
   variant.o:=1;                                                        00202400
   variant.m:=1;                                                        00202500
   i:=variant.l;     {illegal}                                          00202600
   writeln(' ERROR NOT DETECTED...6.4.3.3-8')                           00202700
end.                                                                    00202800
```

```
{TEST 6.4.3.3-9, CLASS=QUALITY}

{ Note this program relies on the compiler deviating for tests
  6.4.3.3-5 to 6.4.3.3-8.
  If the compiler conforms for these tests, this program will
  not compile/run.
  The method of storage for fields of variants may differ,
  depending on the method of definition.
  Programmers should not rely on the VALUES of fields under
  one variant still being accessible from another. However, the
  relationships between the two variants in this example may be
  determined by the output of the program. }

program t6p4p3p3d9(output);
type
   two = (a,b);
var
   variant : record
               case tagfield : two of
                  a: (i,j,k : integer);
                  b: (l : integer;
                      m : integer;
                      n : integer)
             end;
begin
   variant.tagfield:=a;
   variant.i:=1;
   variant.j:=2;
   variant.k:=3;
   variant.tagfield:=b;
   if (variant.l=1) and (variant.m=2) then
      writeln(' EXACT CORRELATION— I:L  J:M  K:N')
   else
   if (variant.l=3) and (variant.m=2) then
      writeln(' REVERSE CORRELATION — I:N  J:M  K:L')
   else
      writeln(' UNKNOWN CORRELATION - lmn are:',
              variant.l,variant.m,variant.n)
end.
```

```
{TEST 6.4.3.3-10, CLASS=CONFORMANCE}

{ The Pascal Standard states that case constants must be distinct,
  and are of an ordinal type which is compatible with the
  tag-field.
  This program tests to see if the compiler will permit case
  constants outside the tag-field subrange - it should .
  The compiler passes if the program runs.
  A warning might be appropriate, however, as fields outside the
  tagfield subrange are not accessible. }

program t6p4p3p3d10(output);
type
   a = 0..3;
   b = record
         case c:a of
            0: (d:array[1..2] of boolean);
            1: (e:array[1..3] of boolean);
            2: (f:array[1..4] of boolean);
            3: (g:array[1..5] of boolean);
            4: (h:array[1..6] of boolean)
       end;
begin
   writeln(' PASS...6.4.3.3-10')
end.


{TEST 6.4.3.3-11, CLASS=DEVIANCE}

{ This program is similar to 6.4.3.3-3, except here, the
  empty record is assigned a value. This should not be possible.
  The program conforms if the program does not compile or run. }

program t6p4p3p3d11(output);
type
   statuskind  = (defined,undefined);
   emptykind   = record end;
var
   empty : emptykind;
   number: record
             case status:statuskind of
                defined  : (i : integer);
                undefined: (e : emptykind)
             end;
begin
   with number do
   begin
      status:=undefined;
      e:=666
   end;
   writeln(' PASS...6.4.3.3-11')
end.
```

```
{TEST 6.4.3.3-12, CLASS=ERRORHANDLING}                                  00211800
                                                                        00211900
{ This program is similar to 6.4.3.3-3, except here                     00212000
  an error is caused by assigning the undefined value                   00212100
  of the variable empty to the field e.                                 00212200
  This error should be detected. }                                      00212300
                                                                        00212400
program t6p4p3p3d12(output);                                            00212500
type                                                                    00212600
   statuskind  = (defined,undefined);                                   00212700
   emptykind   = record end;                                            00212800
var                                                                     00212900
   empty : emptykind;                                                   00213000
   number: record                                                       00213100
            case status:statuskind of                                   00213200
               defined  : (i : integer);                                00213300
               undefined: (e : emptykind)                               00213400
            end;                                                        00213500
begin                                                                   00213600
   with number do                                                       00213700
   begin                                                                00213800
      status:=undefined;                                                00213900
      e:=empty       { undefined despite being empty }                  00214000
   end;                                                                 00214100
   writeln(' PASS...6.4.3.3-12')                                        00214200
end.                                                                    00214300


{TEST 6.4.3.3-13, CLASS=CONFORMANCE}                                    00214400
                                                                        00214500
{ This test checks that nested variants are allowed                     00214600
  with the appropriate syntax. The compiler fails if the                00214700
  program does not compile and print PASS. }                            00214800
                                                                        00214900
program t6p4p3p3d13(output);                                            00215000
type                                                                    00215100
   a=record                                                             00215200
       case b:boolean of                                                00215300
       true: (c:char);                                                  00215400
       false: (case d:boolean of                                        00215500
              true: (e:char);                                           00215600
              false: (f:integer))                                       00215700
     end;                                                               00215800
var                                                                     00215900
   g:a;                                                                 00216000
begin                                                                   00216100
   g.b:=false;                                                          00216200
   g.d:=false;                                                          00216300
   g.f:=1;                                                              00216400
   writeln(' PASS...6.4.3.3-13, VARIANTS')                              00216500
end.                                                                    00216600
```

```
{TEST 6.4.3.4-1, CLASS=CONFORMANCE}                                     00216700
                                                                        00216800
{ This program simply tests that set types as described in the          00216900
  Pascal Standard are permitted.                                        00217000
  The compiler fails if the program will not compile. }                 00217100
                                                                        00217200
program t6p4p3p4d1(output);                                             00217300
type                                                                    00217400
   colour   = (red,blue,pink,green,yellow);                             00217500
   setone   = set of colour;                                            00217600
   settwo   = set of blue..green;                                       00217700
   setthree = set of boolean;                                           00217800
   setfour  = set of 1..10;                                             00217900
   setfive  = set of 0..3;                                              00218000
   setsix   = set of (heart,diamond,spade,club);                        00218100
begin                                                                   00218200
   writeln(' PASS...6.4.3.4-1')                                         00218300
end.                                                                    00218400


{TEST 6.4.3.4-2, CLASS=IMPLEMENTATIONDEFINED}                           00218500
                                                                        00218600
{ This program tests if a set of char is permitted by the               00218700
  compiler. }                                                           00218800
                                                                        00218900
program t6p4p3p4d2(output);                                             00219000
var                                                                     00219100
   s : set of char;                                                     00219200
begin                                                                   00219300
   s:=[';',' ','9','z'];                                                00219400
   if ([';',' ','9','z'] <= s) then                                     00219500
      writeln(' IMPLEMENTATION ALLOWS SET OF CHAR')                     00219600
   else                                                                 00219700
      writeln(' IMPLEMENTATION DOES NOT ALLOW SET OF CHAR')             00219800
end.                                                                    00219900


{TEST 6.4.3.4-3, CLASS=DEVIANCE}                                        00220000
                                                                        00220100
{ The Pascal Standard states that the base-type of the range            00220200
  of a set must be an ordinal-type. This should eliminate sets with     00220300
  real and structured ranges. Some compilers may allow these and        00220400
  hence will deviate for those cases not flagged as errors. }           00220500
                                                                        00220600
program t6p4p3p4d3(output);                                             00220700
type                                                                    00220800
   legalset = set of 1..3;                                              00220900
   urray    = array[1..4] of integer;                                   00221000
   setone   = set of real;                    { case 1 }                00221100
   settwo   = set of record a : 0..3 end;     { case 2 }                00221200
   setthree = set of array[1..5] of real;     { case 3 }                00221300
   setfour  = set of urray;                   { case 4 }                00221400
   setfive  = set of legalset;                { case 5 }                00221500
   setsix   = set of set of 1..4;             { case 6 }                00221600
begin                                                                   00221700
   writeln(' DEVIATES...6.4.3.4-3')                                     00221800
end.                                                                    00221900
```

```
{TEST 6.4.3.4-4, CLASS=IMPLEMENTATIONDEFINED}

{ The Pascal Standard states that the largest and smallest values
  permitted in the base-type of a set-type are implementation
  defined.
  The size of the base-type permitted may be determined by
  examining which of the cases below are accepted by the compiler. }

program t6p4p3p4d4(output);
type
   setone   = set of -1..+1;
   settwo   = set of char;
   setthree = set of 0..1000;
   setfour  = set of 0..10;
   setfive  = set of 0..20;
   setsix   = set of 0..30;
   setseven = set of 0..40;
   seteight = set of 0..50;
   setnine  = set of 0..60;
   setten   = set of 0..70;
var
   s : setthree;
begin
   s:=[1000];
   writeln(' IMPLEMENTATIONDEFINED...6.4.3.4-4 -->',
           'GOOD IMPLEMENTATION OF SETS')
end.
```

```
{TEST 6.4.3.4-5, CLASS=QUALITY}

{ This test is an implementation of Warshall's algorithm
  in Pascal. It serves to give a program which can be used both
  to time a simple procedure using sets, and which can measure
  the space requirements. In both cases the measurements of the
  procedure Warshall are to be compared.

  By way of comparison, the Tasmanian compiler on the Burroughs
  B6700 yielded
      space = 143 bytes ( 6864 bits)
      time = 0.816461 seconds
   }
program t6p4p3p4d5(output);
const
   size = 79;   {array is (size+1) by (size+1) square}
   words = 4;   {size div 16}
   bitsperword = 16;   {assume everyone allows this}
   bitsminus1 = 15;   {bitsperword-1}
type
   btype = array [0..size] of array [0..words]
         of set of 0..bitsminus1;
var
   seed:integer;
   t1,t2:real;
   original,closure:btype;

function generate:integer;
begin
   seed:=57*seed+1;
   generate := (seed mod (..+1));
   seed:=seed mod 571
end;   {of generate}

procedure fill(var a:btype; p:integer);
var
   i:0..size;
   j:0..bitsminus1;
   k,l:0..maxint;
begin
   for j:=0 to words do a[0][j]:=[];
   for i:=1 to size do a[i]:=a[0];
   for k:=1 to p do begin
      i:=generate;
      l:=generate;
      j:=l div bitsperword;
      a[i][j] := a[i][j]+[l mod bitsperword)]
   end
end;   {of fill}

procedure print(var b:btype);
var
   i,j:0..size;
begin
   for i:=0 to size do begin
      write(' ');
      for j:=0 to size do begin
         if (j mod bitsperword) in b[i][j div bitsperword] then
            write('+')
         else
            write('-')
```

```
      end;                                                              00230800
      writeln                                                           00230900
   end                                                                  00231000
end;   {of print}                                                      00231100
                                                                        00231200
procedure warshallsalgorithm(var a,b:btype);                            00231300
   {examine the code to see how many bytes of 8-bits are required}      00231400
var                                                                     00231500
   i,j:0..size;                                                         00231600
   k:0..words;                                                          00231700
begin                                                                   00231800
   b:=a;                                                                00231900
   for i:=0 to size do                                                  00232000
      for j:=0 to size do                                               00232100
         if (i mod bitsperword) in b[j][i div bitsperword] then         00232200
            for k:=0 to words do                                        00232300
               b[j][k] := b[j][k]+b[i][k];                              00232400
end;   {of warshallsalgorithm}                                         00232500
                                                                        00232600
begin    {of main program}                                              00232700
   seed:=1;                                                             00232800
   fill(original,125);                                                  00232900
   t1:=processtime;   {ie begin timing however you do it}               00233000
   warshallsalgorithm(original,closure);                                00233100
   t2:=processtime;   {ie stop timing}                                  00233200
   writeln(' TIME=',t2-t1);                                             00233300
   writeln(' ORIGINAL MATRIX');                                         00233400
   print(original);                                                     00233500
   writeln(' TRANSITIVE CLOSURE');                                      00233600
   print(closure)                                                       00233700
end.                                                                    00233800


{TEST 6.4.3.5-1, CLASS=CONFORMANCE}                                     00233900
                                                                        00234000
{ A file-type is a structured type consisting of a sequence of          00234100
  components which are all one type. All cases in this program          00234200
  should pass.                                                          00234300
  The compiler fails if one or more cases are rejected. }               00234400
                                                                        00234500
program t6p4p3p5d1(output);                                             00234600
type                                                                    00234700
   i = integer;                                                         00234800
var                                                                     00234900
   ptrtoi: ↑i;                                                          00235000
   file1 : file of char;                                                00235100
   file2 : file of real;                                                00235200
   file3 : file of                                                      00235300
            record                                                      00235400
               a : integer;                                             00235500
               b : boolean                                              00235600
            end;                                                        00235700
   file4 : file of set of (red,blue,green,purple);                      00235800
   file5 : file of ptrtoi;                                              00235900
begin                                                                   00236000
   writeln(' PASS...6.4.3.5-1')                                         00236100
end.                                                                    00236200
```

```
{TEST 6.4.3.5-2, CLASS=CONFORMANCE}                                     00236300
                                                                        00236400
{ The Pascal Standard provides for a predefined filetype, type          00236500
  TEXT. Variables of type TEXT are called TEXT FILES. This program      00236600
  tests that such a type is permitted and that the type adheres         00236700
  to the structure laid down in the Standard. The compiler fails        00236800
  if the program will not compile and run. }                            00236900
                                                                        00237000
program t6p4p3p5d2(output);                                             00237100
var                                                                     00237200
  filel : text;                                                         00237300
  chare : char;                                                         00237400
procedure ahaa;                                                         00237500
begin                                                                   00237600
   writeln(' FAIL...6.4.3.5-2')                                         00237700
end;                                                                    00237800
                                                                        00237900
begin                                                                   00238000
   rewrite(filel);                                                      00238100
   writeln(filel);                  { no characters but a linemarker}   00238200
   writeln(filel,'ABC');            { characters and linemarker}        00238300
   reset(filel);                                                        00238400
   if eoln(filel) then get(filel)                                       00238500
   else ahaa;                                                           00238600
   if filel↑='A' then get(filel)                                        00238700
   else ahaa;                                                           00238800
   if filel↑='B' then get(filel)                                        00238900
   else ahaa;                                                           00239000
   if filel↑='C' then get(filel)                                        00239100
   else ahaa;                                                           00239200
   if eoln(filel) and (filel↑=' ') then get(filel)                      00239300
   else ahaa;                                                           00239400
   if eof(filel) then                                                   00239500
      writeln(' PASS...6.4.3.5-2')                                      00239600
   else ahaa                                                            00239700
end.                                                                    00239800


{TEST 6.4.3.5-3, CLASS=CONFORMANCE}                                     00239900
                                                                        00240000
{ This program tests if an end of line marker is inserted at the        00240100
  end of the line, if not explicitly done in the program.               00240200
  The structure of a text file requires a closing linemarker.           00240300
  Conforming compilers will either insert the linemarker, or            00240400
  report a run-time error. }                                            00240500
                                                                        00240600
program t6p4p3p5d3(output);                                             00240700
var                                                                     00240800
   filel : text;                                                        00240900
   chare : char;                                                        00241000
begin                                                                   00241100
   rewrite(filel);                                                      00241200
   write(filel,'A');                                                    00241300
   reset(filel);                                                        00241400
   get(filel);                                                          00241500
   if eoln(filel) then writeln(' PASS...6.4.3.5-3')                     00241600
                  else writeln(' FAIL...6.4.3.5-3')                     00241700
end.                                                                    00241800
```

```
{TEST 6.4.3.5-4, CLASS=CONFORMANCE}                                    00241900
                                                                       00242000
{ This program tests if an end-of-line marker is inserted at           00242100
  the end of the line on the predefined file output, if                00242200
  not explicitly done in the program (i.e. is the buffer               00242300
  flushed). See also test 6.4.3.5-3. }                                 00242400
                                                                       00242500
program t6p4p3p5d4(output);                                            00242600
begin                                                                  00242700
   write(' PASS...6.4.3.5-4')                                          00242800
end.                                                                   00242900


{TEST 6.4.4-1, CLASS=CONFORMANCE}                                      00243000
                                                                       00243100
{ This program simply tests that pointer types as described in the     00243200
  Pascal Standard are permitted. }                                     00243300
                                                                       00243400
program t6p4p4d1(output);                                              00243500
type                                                                   00243600
   sett     = set of 1..2;                                             00243700
   urray    = array[1..3] of integer;                                  00243800
   rekord   = record                                                   00243900
                a : integer;                                           00244000
                b : boolean                                            00244100
              end;                                                     00244200
   ptr9     = ↑sett;                                                   00244300
   pureptr  = ↑pureptr;                                                00244400
var                                                                    00244500
   ptr1  : ↑integer;                                                   00244600
   ptr2  : ↑real;                                                      00244700
   ptr3  : ↑boolean;                                                   00244800
   ptr4  : ↑sett;                                                      00244900
   ptr5  : ↑urray;                                                     00245000
   ptr6  : ↑rekord;                                                    00245100
   ptr7  : pureptr;                                                    00245200
   ptr8  : ptr9;                                                       00245300
begin                                                                  00245400
   new(ptr1);                                                          00245500
   new(ptr2);                                                          00245600
   new(ptr3);                                                          00245700
   new(ptr4);                                                          00245800
   new(ptr5);                                                          00245900
   new(ptr6);                                                          00246000
   new(ptr7);                                                          00246100
   new(ptr8);                                                          00246200
   writeln(' PASS...6.4.4-1')                                          00246300
end.                                                                   00246400
```

```
{TEST 6.4.4-2, CLASS=DEVIANCE}                                         00246500
                                                                       00246600
{ This program tests the diagnostic that should be produced by         00246700
  the compiler if the type to which a pointer points is not            00246800
  found. }                                                             00246900
                                                                       00247000
program t6p4p4d2(output);                                              00247100
var                                                                    00247200
   pointer1 : ↑real;                                                   00247300
   pointer2 : ↑rekord;                                                 00247400
begin                                                                  00247500
   new(pointer1);                                                      00247600
   pointer1:=nil;                                                      00247700
   new(pointer2);                                                      00247800
   pointer2:=nil;                                                      00247900
   writeln(' DEVIATES...6.4.4-2')                                      00248000
end.                                                                   00248100


{TEST 6.4.4-3, CLASS=DEVIANCE}                                         00248200
                                                                       00248300
{ Pointers to items in the stack are not allowed.                      00248400
  The ↑ symbol is not permitted to act as an operator                  00248500
  giving the reference to a variable. The compiler deviates            00248600
  if the program compiles and prints DEVIATES. }                       00248700
                                                                       00248800
program t6p4p4d3(output);                                              00248900
var                                                                    00249000
   p: ↑integer;                                                        00249100
   x:integer;                                                          00249200
begin                                                                  00249300
   x:=10;                                                              00249400
   p:=↑x;                                                              00249500
   writeln(' DEVIATES...6.4.4-3, POINTER')                             00249600
end.                                                                   00249700
```

```
{TEST 6.4.5-1, CLASS=CONFORMANCE}                                        00249800
                                                                         00249900
{ The Pascal Standard states that types designated at two or more        00250000
  different places in the program text are identical if the same         00250100
  type identifier is used at these places, or if different identifiers   00250200
  are used which have been defined to be equivalent to each other.       00250300
  This program simply tests that the compiler conforms to the            00250400
  Standard's  description of identity.                                   00250500
  The compiler fails if the program does not compile. }                  00250600
                                                                         00250700
program t6p4p5d1(output);                                                00250800
type                                                                     00250900
   t1 = array[1..5] of boolean;                                          00251000
   t2 = t1;                                                              00251100
   t3 = t2;                                                              00251200
var                                                                      00251300
   a : t1;                                                               00251400
   b : t2;                                                               00251500
   c : t3;                                                               00251600
procedure identical(var a : t1; var b : t2; var c : t3);                 00251700
begin                                                                    00251800
   a[1]:=true;                                                           00251900
   b[1]:=false;                                                          00252000
   c[1]:=true                                                            00252100
end;                                                                     00252200
                                                                         00252300
begin                                                                    00252400
   a[1]:=true;                                                           00252500
   b[1]:=false;                                                          00252600
   c[1]:=false;                                                          00252700
   identical(a,b,c);                                                     00252800
   identical(c,a,b);                                                     00252900
   identical(b,c,a);                                                     00253000
   writeln(' PASS...6.4.5-1')                                            00253100
end.                                                                     00253200
```

```
{TEST 6.4.5-2, CLASS=DEVIANCE}                                           00253300
                                                                         00253400
{ This program simply tests that the compiler does not deviate           00253500
  from the Standard in the case of subranges of the same host            00253600
  being treated as identical.                                            00253700
  The program should fail to compile/execute if the compiler             00253800
  conforms. }                                                            00253900
                                                                         00254000
program t6p4p5d2(output);                                                00254100
type                                                                     00254200
   colour = (red,pink,orange,yellow,green,blue);                         00254300
   subone = red..yellow;                                                 00254400
   subtwo = pink..blue;                                                  00254500
var                                                                      00254600
   colour1 : subone;                                                     00254700
   colour2 : subtwo;                                                     00254800
                                                                         00254900
procedure test(var col1 : subone);                                       00255000
begin                                                                    00255100
   writeln(' DEVIATES...6.4.5-2')                                        00255200
end;                                                                     00255300
                                                                         00255400
begin                                                                    00255500
   { Although colour1 and colour2 are compatible (i.e. subone and        00255600
     subtwo are compatible), they are not identical, and the call        00255700
     to TEST should fail. }                                              00255800
   colour2:=pink;                                                        00255900
   test(colour2)                                                         00256000
end.                                                                     00256100


{TEST 6.4.5-3, CLASS=DEVIANCE}                                           00256200
                                                                         00256300
{ This test is similar to 6.4.5-2, except that deviance in the           00256400
  case of arrays is tested.                                              00256500
  The program should not compile/execute if the compiler                 00256600
  conforms. }                                                            00256700
                                                                         00256800
program t6p4p5d3(output);                                                00256900
type                                                                     00257000
   urrayone = array[1..10] of char;                                      00257100
   urraytwo = array[1..10] of char;                                      00257200
var                                                                      00257300
   arrayone : urrayone;                                                  00257400
   arraytwo : urraytwo;                                                  00257500
                                                                         00257600
procedure test(var urray : urrayone);                                    00257700
begin                                                                    00257800
   writeln(' DEVIATES...6.4.5-3')                                        00257900
end;                                                                     00258000
                                                                         00258100
begin                                                                    00258200
   { The two arraytypes, urrayone and urraytwo, are not identical        00258300
     and hence the call to TEST should fail. }                           00258400
   test(arraytwo)                                                        00258500
end.                                                                     00258600
```

```
{TEST 6.4.5-4, CLASS=DEVIANCE}

{ This program is similar to 6.4.5-3, except that deviance in the
  case of records is tested.
  The program should fail to compile/execute if the compiler
  conforms. }

program t6p4p5d4(output);
type
   recone = record
               a : integer;
               b : boolean
            end;
   rectwo = record
               c : integer;
               d : boolean
            end;
var
   recordone : recone;
   recordtwo : rectwo;

procedure test(var rec : recone);
begin
   writeln(' DEVIATES...6.4.5-4')
end;

begin
   { Although the two record types are compatible, they
     are not identical, and hence the call to TEST
     should fail. }
   recordtwo.c:=0;
   recordtwo.d:=true;
   test(recordtwo)
end.
```

```
{TEST 6.4.5-5, CLASS=DEVIANCE}

{ Again, this test is similar to 6.4.5-4, except that deviance
  for pointers is tested.
  Although the two pointers in this example point to the same
  type, they are not identical.
  The compiler conforms if the program does not compile/execute. }

program t6p4p5d5(output);
type
   rekord = record
               a : integer;
            end;
   ptrone = ↑rekord;
   ptrtwo = ↑rekord;
var
   ptrtorec    : ptrone;
   ptrtorectoo : ptrtwo;

procedure test(var ptr : ptrone);
begin
   writeln(' DEVIATES...6.4.5-5')
end;

begin
   new(ptrtorectoo);
   ptrtorectoo:=nil;
   test(ptrtorectoo)
end.
```

```
{TEST 6.4.5-6, CLASS=CONFORMANCE}

{ Two types are compatible if they are identical or if one is a
  subrange of the other, or if both are subranges of the same type.
  This program tests these points, but with only subranges of the
  same type having some overlap.
  If the message produced is incomplete, or the program does not
  compile, then the compiler fails.}

program t6p4p5d6(output);
type
   colour   = (red,pink,orange,yellow,green,blue,brown);
   colourtoo= colour;
var
   col1  : colour;
   col2  : colourtoo;
   subcol1  : red..yellow;
   subcol2  : orange..blue;
begin
   col1:=red;
   col2:=red;
   if col1 = col2 then write(' PA');
   subcol1:=red;
   if col1 = subcol1 then write('S');
   subcol1:=yellow;
   subcol2:=yellow;
   if subcol1 = subcol2 then writeln('S...6.4.5-6')
end.
```

```
{TEST 6.4.6-2, CLASS=CONFORMANCE}                                    00286200
                                                                     00286300
{ This test is similar to 6.4.6-1, except that it tests the use      00286400
  of assignment compatibility in actual and formal parameters.       00286500
  The compiler fails if the program does not compile. }              00286600
                                                                     00286700
program t6p4p6d2(output);                                            00286800
type                                                                 00286900
   colour = (red,pink,yellow,green);                                 00287000
   subcol1 = yellow..green;                                          00287100
   subcol2 = set of colour;                                          00287200
   subcol3 = set of pink..green;                                     00287300
var                                                                  00287400
   a       : integer;                                                00287500
   b       : real;                                                   00287600
   colour1 : colour;                                                 00287700
   colour2 : pink..green;                                            00287800
   colour3 : set of colour;                                          00287900
   colour4 : set of yellow..green;                                   00288000
                                                                     00288100
procedure compat(i : integer; j : real;                              00288200
                 col1 : colour; col2 : subcol1;                      00288300
                 col3 : subcol2; col4 : subcol3);                    00288400
begin                                                                00288500
end;                                                                 00288600
                                                                     00288700
begin                                                                00288800
   compat(2,2.4,yellow,yellow,[pink],[pink]);                        00288900
   a:=2;                                                             00289000
   b:=3.1;                                                           00289100
   colour1:=pink;                                                    00289200
   colour2:=green;                                                   00289300
   colour3:=[yellow];                                                00289400
   colour4:=[yellow];                                                00289500
   compat(a,b,colour1,colour2,colour3,colour4);                      00289600
   compat(a,a,colour2,colour2,colour4,colour4);                      00289700
   writeln(' PASS...6.4.6-2')                                        00289800
end.                                                                 00289900
```

```
{TEST 6.4.6-3, CLASS=CONFORMANCE}                                    00290000
                                                                     00290100
{ This program tests a part of 6.5.2.1, that states that an index    00290200
  expression is assignment compatible with the index type            00290300
  specified in the definition of the array type.                     00290400
  The compiler fails if the program does not compile. }              00290500
                                                                     00290600
program t6p4p6d3(output);                                            00290700
type                                                                 00290800
   colour = (red,pink,orange,yellow,green);                          00290900
   intensity = (bright,dull);                                        00291000
var                                                                  00291100
   array1 : array[yellow..green] of boolean;                         00291200
   array2 : array[colour] of intensity;                              00291300
   array3 : array[1..99] of integer;                                 00291400
   colour1 : red..yellow;                                            00291500
   i      : integer;                                                 00291600
begin                                                                00291700
   array1[yellow]:=true;                                             00291800
   colour1:=yellow;                                                  00291900
   array1[colour1]:=false;                                           00292000
   array2[colour1]:=bright;                                          00292100
   array3[1]:=0;                                                     00292200
   i:=2;                                                             00292300
   array3[i*3+2]:=1;                                                 00292400
   writeln(' PASS...6.4.6-3')                                        00292500
end.                                                                 00292600


{TEST 6.4.6-4, CLASS=ERRORHANDLING}                                  00292700
                                                                     00292800
{ The Pascal standard says that if the two types in an assignment    00292900
  compatibility test (T1 and T2) are compatible ordinal types        00293000
  and the value of the expression E which is of type T2 is not in    00293100
  the closed interval specified by the type T1, an error occurs.     00293200
  Does this compiler detect this. }                                  00293300
                                                                     00293400
program t6p4p6d4(output);                                            00293500
type                                                                 00293600
   subrange = 0..5;                                                  00293700
var                                                                  00293800
   i : subrange;                                                     00293900
begin                                                                00294000
   i:=5;                                                             00294100
   i:=i*2;    { error }                                              00294200
   writeln(' ERROR NOT DETECTED...6.4.6-4')                          00294300
end.                                                                 00294400
```

```
{TEST 6.4.6-5, CLASS=ERRORHANDLING}                                    00294500
                                                                       00294600
{ This program is similar to 6.4.6-4, except that parameter            00294700
  assignment compatibility is tested.                                  00294800
  The program causes an error to occur which should be detected. }     00294900
                                                                       00295000
program t6p4p6d5(output);                                              00295100
type                                                                   00295200
   subrange = 0..5;                                                    00295300
var                                                                    00295400
   i : subrange;                                                       00295500
procedure test(a : subrange);                                          00295600
begin                                                                  00295700
   a:=5                                                                00295800
end;                                                                   00295900
                                                                       00296000
begin                                                                  00296100
   i:=5;                                                               00296200
   test(i*2);       { error }                                          00296300
   writeln(' ERROR NOT DETECTED...6.4.6-5')                            00296400
end.                                                                   00296500
```

```
{TEST 6.4.6-6, CLASS=ERRORHANDLING}                                    00296600
                                                                       00296700
{ This program is similar to 6.4.6-4, except that array               00296800
  subscript assignment compatibility is tested.                        00296900
  The program causes an error, which should be detected. }             00297000
                                                                       00297100
program t6p4p6d6(output);                                              00297200
type                                                                   00297300
   colour = (red,pink,orange,yellow,green);                            00297400
var                                                                    00297500
   v     : colour;                                                     00297600
   urray : array[red..orange] of boolean;                              00297700
begin                                                                  00297800
   v:=orange;                                                          00297900
   urray[succ(v)]:=true;       { error }                               00298000
   writeln(' ERROR NOT DETECTED...6.4.6-6')                            00298100
end.                                                                   00298200
```

```
{TEST 6.4.6-7, CLASS=ERRORHANDLING}                                    00298300
                                                                       00298400
{ Similarly for 6.4.6-4, if two types are compatible set types,        00298500
  and any of the members of the set expression E (of type T2)          00298600
  is not in the closed interval specified by the base-type of the      00298700
  type T1, an error occurs.                                            00298800
  Again, does the compiler detect this. }                              00298900
                                                                       00299000
program t6p4p6d7(output);                                              00299100
type                                                                   00299200
   colour = (red,pink,orange,yellow,green,blue);                       00299300
   subone = red..orange;                                               00299400
   subtwo = pink..yellow;                                              00299500
var                                                                    00299600
   setone : set of subone;                                             00299700
   settwo : set of subtwo;                                             00299800
begin                                                                  00299900
   settwo:=[pink,yellow];                                              00300000
   setone:=settwo;                          { should be an error }     00300100
   writeln(' ERROR NOT DETECTED...6.4.6-7')                            00300200
end.                                                                   00300300
```

```
{TEST 6.4.6-8, CLASS=ERRORHANDLING}                                    00300400
                                                                       00300500
{ This test is similar to 6.4.6-7, except that assignment              00300600
  compatibility for sets passed as parameters is tested.               00300700
  The program causes an error which should be detected. }              00300800
                                                                       00300900
program t6p4p6d8(output);                                              00301000
type                                                                   00301100
   colour   = (red,pink,orange,yellow,green,blue);                     00301200
   subone   = red..green;                                              00301300
   settwo   = set of yellow..blue;                                     00301400
var                                                                    00301500
   setone : set of subone;                                             00301600
procedure test(a : settwo);                                            00301700
begin                                                                  00301800
end;                                                                   00301900
                                                                       00302000
begin                                                                  00302100
   setone:=[red,pink,orange];                                          00302200
   test(setone);                                                       00302300
   writeln(' ERROR NOT DETECTED...6.4.6-8')                            00302400
end.                                                                   00302500
```

```
{TEST 6.4.6-9, CLASS=DEVIANCE}

{ The Pascal Standard allows assignment of integers to reals,
  but not reals to integers.
  Does this compiler allow assignment of reals to integers.
  If so, it does not conform to the Standard.
  The compiler conforms if the program does not compile. }

program t6p4p6d9(output);
var
   i : real;
   j : integer;
procedure test(a:integer);
begin
end;

begin
   i:=6.345;
   j:=i;
   test(6.345);
   writeln(' DEVIATES...6.4.6-9')
end.
```

```
{TEST 6.4.6-10, CLASS=DEVIANCE}

{ The Pascal Standard states that the two types T1 and T2
  (in determining assignment compatibility) must neither be a
  a file type nor a structured type with a file component.
  This program tests the first part of this statement.
  The compiler conforms if the program does not compile. }

program t6p4p6d10(output);
var
   file1 : text;
   file2 : text;
begin
   reset(file1);
   rewrite(file2);
   writeln(file1,'ABC');
   file2:=file1;
   writeln(' DEVIATES...6.4.6-10')
end.
```

```
{TEST 6.4.6-11, CLASS=DEVIANCE}

{ This program tests the latter half of the statement in
  6.4.6-10.
  The compiler conforms if the program does not compile. }

program t6p4p6d11(output);
type
   rekord = record
               f : text;
               a : integer
            end;
var
   record1 : rekord;
   record2 : rekord;
begin
   record1.a:=1;
   reset(record1.f);
   rewrite(record2.f);
   writeln(record1.f);
   record2:=record1;;
   writeln(' DEVIATES...6.4.6-11')
end.
```

```
{TEST 6.4.6-12, CLASS=DEVIANCE}

{ The standard specifies that a filetype T2 cannot be
  assignment-compatible with an identical type T1, nor can a
  structure containing such a filetype. This precludes any
  assignments involving files. The compiler deviates if the program
  compiles and prints DEVIATES. }

program t6p4p6d12(output);
var
   f1,f2:text;
begin
   rewrite(f1);
   writeln(f1,' DEVIATES');
   writeln(' DEVIATES...6.4.6-12, FILES');
   f2:=f1;
end.
```

```
{TEST 6.5.1-1, CLASS=CONFORMANCE}                                    00310700
                                                                     00310800
{ Here is included two examples from the Pascal Standard.            00310900
  The first is from section 6.4.7, and consists of legal type        00311000
  declarations. The second is from section 6.5.1, and consists       00311100
  of legal variable declarations.                                    00311200
  The compiler fails if the program does not compile. }              00311300
                                                                     00311400
program t6p5pld1(output);                                            00311500
type                                                                 00311600
   count    = integer;                                               00311700
   range    = integer;                                               00311800
   colour   = (red,yellow,green,blue);                               00311900
   sex      = (male,female);                                         00312000
   year     = 1900..1999;                                            00312100
   shape    = (triangle,rectangle,circle);                           00312200
   card     = array[1..80] of char;                                  00312300
   str      = file of char;                                          00312400
   angle    = real;                                                  00312500
   polar    = record                                                 00312600
                  r : real;                                          00312700
                  theta : angle                                      00312800
              end;                                                   00312900
   person   = ↑ persondetails;                                       00313000
   persondetails = record                                            00313100
            name, firstname : str;                                   00313200
            age : integer;                                           00313300
            married : boolean;                                       00313400
            father,child,sibling : person;                           00313500
            case s:sex of                                            00313600
               male   : (enlisted,bearded : boolean);                00313700
               female : (pregnant : boolean)                         00313800
            end;                                                     00313900
   tape     = file of persondetails;                                 00314000
   intfile  = file of integer;                                       00314100
                                                                     00314200
var                                                                  00314300
   x,y,z    : real;                                                  00314400
   i,j      : integer;                                               00314500
   k        : 0..9;                                                  00314600
   p,q,r    : boolean;                                               00314700
   operator : (plus,minus,times);                                    00314800
   a        : array[0..63] of real;                                  00314900
   c        : colour;                                                00315000
   f        : file of char;                                          00315100
   huel,hue2 : set of colour;                                        00315200
   p1,p2    : person;                                                00315300
   m,m1,m2  : polar;                                                 00315400
   pooltape : array[1..4] of tape;                                   00315500
begin                                                                00315600
   writeln(' PASS...6.5.1-1')                                        00315700
end.                                                                 00315800
```

```
{TEST 6.5.1-2, CLASS=QUALITY}                                        00315900
                                                                     00316000
{ This test checks that long declaration lists are allowed by        00316100
  the compiler. The test may detect a small compiler limit . }       00316200
                                                                     00316300
program t6p5pld2(output);                                            00316400
var                                                                  00316500
   i0,i1,i2,i3,i4,i5,i6,i7,i8,i9,                                    00316600
   i10,i11,i12,i13,i14,i15,i16,i17,i18,i19,                          00316700
   i20,i21,i22,i23,i24,i25,i26,i27,i28,i29,                          00316800
   i30,i31,i32,i33,i34,i35,i36,i37,i38,i39,                          00316900
   i40,i41,i42,i43,i44,i45,i46,i47,i48,i49,                          00317000
   i50,i51,i52,i53,i54,i55,i56,i57,i58,i59,                          00317100
   i60,i61,i62,i63,i64,i65,i66,i67,i68,i69,                          00317200
   i70,i71,i72,i73,i74,i75,i76,i77,i78,i79,                          00317300
   i80,i81,i82,i83,i84,i85,i86,i87,i88,i89,                          00317400
   i90,i91,i92,i93,i94,i95,i96,i97,i98,i99                           00317500
   : integer;                                                        00317600
begin                                                                00317700
  i0:=0; i1 :=1; i2:=2; i3:=3; i4:=4; i5:=5; i6:=6; i7:=7;i8:=8; i9:=9; 00317800
  i10:=i0+1; i11:=i1+1; i12:=i2+1; i13:=i3+1; i14:=i4+1;             00317900
  i15:=i5+1; i16:=i6+1; i17:=i7+1; i18:=i8+1; i19:=i9+1;             00318000
  i20:=i10+i0; i21:=i11+i1; i22:=i12+i2; i23:=i13+i3; i24:=i14+i4;   00318100
  i25:=i15+i5; i26:=i16+i6; i27:=i17+i7; i28:=i18+i8; i29:=i19+i9;   00318200
  i30:=i20+i10; i31:=i21+i11; i32:=i22+i12; i33:=i23+i13; i34:=i24+i14; 00318300
  i35:=i25+i15; i36:=i26+i16; i37:=i27+i17; i38:=i28+i18; i39:=i29+i19; 00318400
  i40:=i30+i20; i41:=i31+i21; i42:=i32+i22; i43:=i33+i23; i44:=i34+i24; 00318500
  i45:=i35+i25; i46:=i36+i26; i47:=i37+i27; i48:=i38+i28; i49:=i39+i29; 00318600
  i50:=i40+i30; i51:=i41+i31; i52:=i42+i32; i53:=i43+i33; i54:=i44+i34; 00318700
  i55:=i45+i35; i56:=i46+i36; i57:=i47+i37; i58:=i48+i38; i59:=i49+i39; 00318800
  i60:=i50+i40; i61:=i51+i41; i62:=i52+i42; i63:=i53+i43; i64:=i54+i44; 00318900
  i65:=i55+i45; i66:=i56+i46; i67:=i57+i47; i68:=i58+i48; i69:=i59+i49; 00319000
  i70:=i60+i50; i71:=i61+i51; i72:=i62+i52; i73:=i63+i53; i74:=i64+i54; 00319100
  i75:=i65+i55; i76:=i66+i56; i77:=i67+i57; i78:=i68+i58; i79:=i69+i59; 00319200
  i80:=i70+i60; i81:=i71+i61; i82:=i72+i62; i83:=i73+i63; i84:=i74+i64; 00319300
  i85:=i75+i65; i86:=i76+i66; i87:=i77+i67; i88:=i78+i68; i89:=i79+i69; 00319400
  i90:=i80+i70; i91:=i81+i71; i92:=i82+i72; i93:=i83+i73; i94:=i84+i74; 00319500
  i95:=i85+i75; i96:=i86+i76; i97:=i87+i77; i98:=i88+i78; i99:=i89+i79; 00319600
  i0:=i90+i91+i92+i93+i94+i95+i96+i97+i98+i99;                       00319700
  if (i0=2815) then                                                  00319800
     writeln(' LONG DECLARATIONS ALLOWED...6.5.1-2')                 00319900
  else                                                               00320000
     writeln(' LONG DECLARATIONS NOT ALLOWED...6.5.1-2');            00320100
end.                                                                 00320200
```

```
{TEST 6.5.3.2-1, CLASS=ERRORHANDLING}

{ This test is similar to 6.4.5-6, except that a two
  dimensional array is used. This may present some problems to
  particular implementations. }

program t6p5p3p2d1(output);
var
   urray : array[1..10,1..10] of integer;
   i     : integer;
begin
   i:=3;
   urray[i*2,i*4]:=0;
   writeln(' ERROR NOT DETECTED...6.5.3.2-1')
end.
```

```
{TEST 6.5.3.2-2, CLASS=CONFORMANCE}

{ This test checks that the two ways of indexing a multi-dimensional
  array are equivalent. The compiler fails if the program does
  not compile and print PASS.}

program t6p5p3p2d2(output);
var
   a:array[1..4,1..4] of integer;
   b:array[1..4] of
      array[1..4] of integer;
   p:packed array [1..4,1..4]of char;
   q:packed array[1..4] of
     packed array [1..4] of char;
   i,j,counter:integer;
begin
   counter:=0;
   for i:= 1 to 4 do
      for j:=1 to 4 do
      begin
         a[i,j] := j;
         b[i,j] := j;
         case j of
         1:
           begin
               p[i,j]:='F';
               q[i,j]:='F';
           end;
         2:
           begin
               p[i,j]:='A';
               q[i,j]:='A';
           end;
         3:
           begin
               p[i,j]:='I';
               q[i,j]:='I';
           end;
         4:
           begin
               p[i,j]:='L';
               q[i,j]:='L';
           end;
         end;
      end;
   for i:=1 to 4 do
      for j:=1 to 4 do
      begin
         if a[i][j] <> a[i,j] then
            counter:=counter+1;
         if b[i][j] <> b[i,j] then
            counter:=counter+1;
         if p[i][j] <> p[i,j] then
            counter:=counter+1;
         if q[i][j] <> q[i,j] then
            counter:=counter+1;
      end;
   if counter=0 then
      writeln(' PASS...6.5.3.2-2, INDEXING')
   else
      writeln(' FAIL...6.5.3.2-2, INDEXING')
```

```
end.                                                                    00327900


{TEST 6.5.3.4-1, CLASS=CONFORMANCE}                                     00328000
                                                                        00328100
{ The Pascal Standard states that the existance of a file               00328200
  variable f with components of type T implies the existence            00328300
  of a buffer variable of type T.                                       00328400
  Only the one component of a file variable determined by the           00328500
  current file position is directly accessible.                         00328600
  The program tests that file buffers may be referenced in this         00328700
  implementation.                                                       00328800
  The compiler fails if the program does not compile. }                 00328900
                                                                        00329000
program t6p5p3p4d1(output);                                             00329100
type                                                                    00329200
   rekord = record                                                      00329300
               urray : array[1..2] of char;                             00329400
               a : integer;                                             00329500
               b : real                                                 00329600
            end;                                                        00329700
var                                                                     00329800
   fyle : file of rekord;                                               00329900
begin                                                                   00330000
   rewrite(fyle);                                                       00330100
   fyle↑.urray[1]:='O';                                                 00330200
   fyle↑.urray[2]:='K';                                                 00330300
   fyle↑.a:=10;                                                         00330400
   fyle↑.b:=2.345;                                                      00330500
   put(fyle);                                                           00330600
   with fyle↑ do                                                        00330700
   begin                                                                00330800
      urray[1]:='O';                                                    00330900
      urray[2]:='K';                                                    00331000
      a:=4;                                                             00331100
      b:=3.456                                                          00331200
   end;                                                                 00331300
   put(fyle);                                                           00331400
   writeln(' PASS...6.5.3.4-1')                                         00331500
end.                                                                    00331600


{TEST 6.5.4-1, CLASS=ERRORHANDLING}                                     00331700
                                                                        00331800
{ The Pascal Standard states that an error occurs if a pointer          00331900
  variable has a value NIL at the time it is dereferenced.              00332000
  This program tests that the error is detected. The diagnostic         00332100
  should be checked for suitability. }                                  00332200
                                                                        00332300
program t6p5p4d1(output);                                               00332400
type                                                                    00332500
   rekord = record                                                      00332600
               a : integer;                                             00332700
               b : boolean                                              00332800
            end;                                                        00332900
var                                                                     00333000
   pointer : ↑rekord;                                                   00333100
begin                                                                   00333200
   pointer:=nil;                                                        00333300
   pointer↑.a:=1;                                                       00333400
   pointer↑.b:=true;                                                    00333500
   writeln(' ERROR NOT DETECTED...6.5.4-1')                             00333600
end.                                                                    00333700


{TEST 6.5.4-2, CLASS=ERRORHANDLING}                                     00333800
                                                                        00333900
{ Similarly to 6.5.4-1, an error occurs if a pointer variable           00334000
  has an undefined value when it is dereferenced. }                     00334100
                                                                        00334200
program t6p5p4d2(output);                                               00334300
type                                                                    00334400
   rekord = record                                                      00334500
               a : integer;                                             00334600
               b : boolean                                              00334700
            end;                                                        00334800
var                                                                     00334900
   pointer : ↑rekord;                                                   00335000
begin                                                                   00335100
   pointer↑.a:=1;                                                       00335200
   pointer↑.b:=true;                                                    00335300
   writeln(' ERROR NOT DETECTED...6.5.4-2')                             00335400
end.                                                                    00335500
```

```
{TEST 6.6.1-7, CLASS=QUALITY}                                                  00350700
                                                                               00350800
{ This test checks that procedures may be nested to 15 levels.                 00350900
  The test may detect a small compiler limit.  The limit may                   00351000
  arise due to failure of a register allocation scheme, a limited              00351100
  reserved size for a display, or a field set aside for lexical                00351200
  level information, or some combination of these. }                           00351300
                                                                               00351400
program t6p6p1d7(output);                                                      00351500
var                                                                            00351600
   i:integer;                                                                  00351700
                                                                               00351800
procedure p1;                                                                  00351900
   procedure p2;                                                               00352000
      procedure p3;                                                            00352100
         procedure p4;                                                         00352200
            procedure p5;                                                      00352300
               procedure p6;                                                   00352400
                  procedure p7;                                                00352500
                     procedure p8;                                             00352600
                        procedure p9;                                          00352700
                           procedure p10;                                      00352800
                              procedure p11;                                   00352900
                                 procedure p12;                                00353000
                                    procedure p13;                             00353100
                                       procedure p14;                          00353200
                                          procedure p15;                       00353300
                                          begin                                00353400
                                             i:=i+1;                           00353500
                                          end;                                 00353600
                                       begin                                   00353700
                                          p15                                  00353800
                                       end;                                    00353900
                                    begin                                      00354000
                                       p14                                     00354100
                                    end;                                       00354200
                                 begin                                         00354300
                                    p13                                        00354400
                                 end;                                          00354500
                              begin                                            00354600
                                 p12                                           00354700
                              end;                                             00354800
                           begin                                               00354900
                              p11                                              00355000
                           end;                                                00355100
                        begin                                                  00355200
                           p10                                                 00355300
                        end;                                                   00355400
                     begin                                                     00355500
                        p9                                                     00355600
                     end;                                                      00355700
                  begin                                                        00355800
                     p8                                                        00355900
                  end;                                                         00356000
               begin                                                           00356100
                  p7                                                           00356200
               end;                                                            00356300
            begin                                                              00356400
               p6                                                              00356500
            end;                                                               00356600
         begin                                                                 00356700
               p5                                                              00356800
            end;                                                               00356900
         begin                                                                 00357000
            p4                                                                 00357100
         end;                                                                  00357200
      begin                                                                    00357300
         p3                                                                    00357400
      end;                                                                     00357500
   begin                                                                       00357600
      p2                                                                       00357700
   end;                                                                        00357800
                                                                               00357900
   begin                                                                       00358000
      i:=0;                                                                    00358100
      p1;                                                                      00358200
      writeln(' NESTED PROCEDURES TO 15 LEVELS IMPLEMENTED...6.6.1-7');        00358300
   end.                                                                        00358400
```

```
{TEST 6.6.2-1, CLASS=CONFORMANCE}

{ This program simply tests the syntax for functions as defined
  by the Pascal Standard.
  The compiler fails if the program does not compile. }

program t6p6p2d1(output);
var
   a ,
   twopisquared : real;
   b : integer;

function power(x : real; y : integer):real;  { y>=0 }
var
   w,z : real;
   i : 0..maxint;
begin
   w:=x;
   z:=1;
   i:=y;
   while i > 0 do
   begin
      { z*(w tothepower i)=x tothepower y }
      if odd(i) then z:=z*w;
      i:=i div 2;
      w:=sqr(w)
   end;
   { z=x tothepower y }
   power:=z
end;

function twopi : real;
begin
   twopi:=6.283185
end;

begin
   a:=twopi;
   b:=2;
   twopisquared:=power(a,b);
   writeln(' PASS...6.6.2-1')
end.
```

```
{TEST 6.6.2-2, CLASS=CONFORMANCE}

{ Similarly to 6.6.1-2, functions may be declared as forward.
  This program tests that forward declaration and recursion in
  functions is permitted.
  The compiler fails if the program does not compile. }

program t6p6p2d2(output);
var
   c : integer;
function one(a : integer) : integer;
   forward;

function two(b : integer) : integer;
var
   x : integer;
begin
   x:=b+1;
   x:=one(x);
   two:=x
end;

function one;
var
   y : integer;
begin
   y:=a+1;
   if y=1 then y:=two(y);
   one:=y
end;

begin
   c:=0;
   c:=one(c);
   if c = 3 then
      writeln(' PASS...6.6.2-2')
end.
```

```
{TEST 6.6.2-3, CLASS=CONFORMANCE}

{ The Pascal Standard specifies that the result type of a function
  can only be a simple type or a pointer type.
  This program checks that the simple types and pointer types are
  permitted.
  The compiler fails if the program does not compile. }

program t6p6p2d3(output);
type
   subrange = 0..3;
   enumerated = (red,yellow,green);
   rectype = record
               a : integer
             end;
   ptrtype = ↑rectype;
var
   a : real;
   b : integer;
   c : boolean;
   d : subrange;
   e : enumerated;
   f : char;
   g : ptrtype;

function one : real;
begin
   one:=2.63
end;
function two : integer;
begin
   two:=2
end;
function three : boolean;
begin
   three:=false
end;
function four : subrange;
begin
   four:=2
end;
function five : enumerated;
begin
   five:=yellow
end;
function six : char;
begin
   six:='6'
end;
function seven : ptrtype;
begin
   seven:=nil
end;

begin
   a:=one;
   b:=two;
   c:=three;
   d:=four;
   e:=five;
   f:=six;
   g:=seven;
   writeln(' PASS...6.6.2-3')
end.


{TEST 6.6.2-4, CLASS=DEVIANCE}

{ This program tests the compilers actions when the type of
  result returned by a function is not a simple type.
  All the cases should be rejected by the compiler if it
  conforms to the Standard. }

program t6p6p2d4(output);
type
   wrekord = record
               a : integer;
               b : boolean
             end;
   sett    = set of 0..3;
   urray   = array[1..3] of char;
var
   record1 : wrekord;
   set1    : sett;
   array1  : urray;

function one : sett;
begin
   one:=[0..3]
end;

function two : urray;
begin
   two:='ABC'
end;

function three : wrekord;
var
   rekord : wrekord;
begin
   rekord.a:=1;
   rekord.b:=true;
   three:=rekord
end;

begin
   record1:=one;
   set1:=two;
   array1:=three;
   writeln(' DEVIATES...6.6.2-4')
end.
```

```
{TEST 6.6.2-5, CLASS=DEVIANCE}                                          00377300

{ The Pascal Standard specifies that at least one assignment            00377400
  statement which assigns a value to the function identifier            00377500
  must occur in the function block.                                     00377600
  Does the compiler permit a function declaration with                  00377700
  no assignment to the function identifier?                             00377800
  The compiler deviates if it does. }                                   00377900

program t6p6p2d5(output);                                               00378000
var                                                                     00378100
   a : integer;                                                         00378200
function illegal(var b : integer) : integer;                            00378300
var                                                                     00378400
   x : integer;                                                         00378500
begin                                                                   00378600
   x:=b*2                                                               00378700
end;                                                                    00378800

begin                                                                   00378900
   a:=2;                                                                00379000
   a:=illegal(a);                                                       00379100
   writeln(' DEVIATES...6.6.2-5')                                       00379200
end.                                                                    00379300


{TEST 6.6.2-6, CLASS=ERRORHANDLING}                                     00379400

{ The Pascal Standard states that the result of a function will         00379500
  be the last value assigned to its identifier. If no assignment        00379600
  occurs then the result is undefined.                                  00379700
  This program contains a function with an assignment to its            00379800
  identifier, however the assignment is never executed. An error        00379900
  should occur during execution. }                                      00380000

program t6p6p2d6(output);                                               00380100
var                                                                     00380200
   radius ,                                                             00380300
   circlearea : real;                                                   00380400
function area(a : real) : real;                                         00380500
var                                                                     00380600
   x : real;                                                            00380700
begin                                                                   00380800
   if a > 0 then x:=3.1415926*a*a                                       00380900
            else area:=0                                                00381000
end;                                                                    00381100

begin                                                                   00381200
   radius:=2;                                                           00381300
   circlearea:=area(radius);                                            00381400
   writeln(' ERROR NOT DETECTED...6.6.2-6')                             00381500
end.                                                                    00381600
                                                                        00381700
{TEST 6.6.2-7, CLASS=CONFORMANCE}                                       00381800

{ This test checks that functions are not prohibited from altering      00381900
  their environment (ie. side effects). Though side effects are         00382000
  generally not to be encouraged, they are part of standard Pascal      00382100
  and do have genuine uses. Functions with side effect occur            00382200
  elsewhere in the validation suite. }                                  00382300
                                                                        00382400
program t6p6p2d7(output);                                               00382500
type                                                                    00382600
   ptrtochar = ↑char;                                                   00382700
var                                                                     00382800
   c1,c2,c3,dummy:char;                                                 00382900
   p1,p2:ptrtochar;                                                     00383000
                                                                        00383100
function testa(ptr:ptrtochar):char;                                     00383200
   {sneakiest, uses pointers}                                           00383300
var                                                                     00383400
   pp:ptrtochar;                                                        00383500
begin                                                                   00383600
   pp:=ptr;                                                             00383700
   pp↑ := 'P';                                                          00383800
   testa:='1'                                                           00383900
end;                                                                    00384000

procedure assign;                                                       00384100
   {used by testb}                                                      00384200
begin                                                                   00384300
   c1:='A'                                                              00384400
end;                                                                    00384500

function testb:char;                                                    00384600
   {sneaky, calls a procedure}                                          00384700
begin                                                                   00384800
   assign;                                                              00384900
   testb:='2'                                                           00385000
end;                                                                    00385100

function testc:char;                                                    00385200
   {blatantly changes the environment via write}                        00385300
begin                                                                   00385400
   write(' ',p1↑,c1,c2,c3,p2↑);                                         00385500
   testc:='6'                                                           00385600
end;                                                                    00385700

function testd:ptrtochar;                                               00385800
   {blatantly sneaky: modifying the environment via new                 00385900
      and then passing it out}                                          00386000
var                                                                     00386100
   pp:ptrtochar;                                                        00386200
begin                                                                   00386300
   new(pp);                                                             00386400
   pp↑:='.';                                                            00386500
   testd:=pp                                                            00386600
end;                                                                    00386700

function teste:char;                                                    00386800
   {the most used side effect:global access}                            00386900
begin                                                                   00387000
   c2:='S';                                                             00387100
   teste:='3'                                                           00387200
```

```
end;                                                                    00388400
                                                                        00388500
function testf(var c:char):char;                                        00388600
   {straightforward}                                                    00388700
begin                                                                   00388800
   c:='S';                                                              00388900
   testf:='4'                                                           00389000
end;                                                                    00389100
                                                                        00389200
begin {of main program}                                                 00389300
   new(p1);                                                             00389400
   p1^:='F'; c1:='A'; c2:='I'; c3:='L';                                 00389500
   p2:=nil;                                                             00389600
      {which defines all variables}                                     00389700
   dummy:=testa(p1);                                                    00389800
   dummy:=testb;                                                        00389900
   dummy:=teste;                                                        00390000
   dummy:=testf(c3);                                                    00390100
   p2:=testd;                                                           00390200
   dummy:=testc;                                                        00390300
   writeln('..6.6.2-7, ENVIRONMENT')                                    00390400
end.                                                                    00390500
```

```
{TEST 6.6.3.1-1, CLASS=CONFORMANCE}                                     00390600
                                                                        00390700
{ This program tests that parameters as described by the Pascal         00390800
  Standard are permitted by the compiler, especially long               00390900
  identifier lists. A parameter list with 30 elements is thought        00391000
  long enough to test most applications using procedure/function        00391100
  parameter lists. This test occurs elsewhere in the suite, but         00391200
  is included here for consistency.                                     00391300
  The compiler fails if the program does not compile. }                 00391400
                                                                        00391500
program t6p6p3p1d1(output);                                             00391600
type                                                                    00391700
   colour   = (red,orange,yellow,green,blue,brown);                     00391800
   subrange = red..blue;                                                00391900
   rekord   = record                                                    00392000
                a : integer                                             00392100
              end;                                                      00392200
   ptrtype  = ^rekord;                                                  00392300
var                                                                     00392400
   a,b,c,d,e,f,g,h,i,j,                                                 00392500
   k,l,m,n,o,p,q,r,s,t : integer;                                       00392600
   colone : colour;                                                     00392700
   coltwo : colour;                                                     00392800
   colthree : colour;                                                   00392900
   u,v,w,x : real;                                                      00393000
   y,z : boolean;                                                       00393100
   ptr : ptrtype;                                                       00393200
                                                                        00393300
procedure testone(a1,b1,c1,d1,e1,f1,g1,h1,i1,j1,k1,                     00393400
                  l1,m1,n1,o1,p1,q1,r1,s1,t1 : integer;                 00393500
                  colourone : subrange;                                 00393600
                  colourtwo,colourthree : colour;                       00393700
                  u1,v1,w1,x1 : real;                                   00393800
                  y1,z1 : boolean;                                      00393900
                  ptr : ptrtype);                                       00394000
begin                                                                   00394100
   write(' PASS')                                                       00394200
end;                                                                    00394300
procedure testtwo(var a1,b1,c1,d1,e1,f1,g1,h1,i1,j1,k1,                 00394400
                  l1,m1,n1,o1,p1,q1,r1,s1,t1 : integer;                 00394500
                  var colourone : subrange;                             00394600
                  var colourtwo,colourthree : colour;                   00394700
                  var u1,v1,w1,x1 : real;                               00394800
                  var y1,z1 : boolean;                                  00394900
                  var ptr : ptrtype);                                   00395000
begin                                                                   00395100
   writeln('...6.6.3.1-1')                                             00395200
end;                                                                    00395300
                                                                        00395400
begin                                                                   00395500
   a:=0; b:=0; c:=0; d:=0; e:=0; f:=0; g:=0;                            00395600
   h:=0; i:=0; j:=0; k:=0; l:=0; m:=0; n:=0;                            00395700
   o:=0; p:=0; q:=0; r:=0; s:=0; t:=0;                                  00395800
   colone:=orange;                                                      00395900
   coltwo:=brown;                                                       00396000
   colthree:=red;                                                       00396100
   u:=0; v:=0; w:=0; x:=0;                                              00396200
   y:=true;                                                             00396300
   z:=false;                                                            00396400
   new(ptr);                                                            00396500
   testone(a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,                     00396600
```

```
          colone,coltwo,colthree,u,v,w,x,y,z,ptr);                00396700
   testtwo(a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,                00396800
          colone,coltwo,colthree,u,v,w,x,y,z,ptr);                00396900
end.                                                              00397000


{TEST 6.6.3.1-2, CLASS=CONFORMANCE}                               00397100
                                                                  00397200
{ This program is similar to 6.6.3.1-1, except that set,          00397300
  record and array parameter lists are tested.                    00397400
  The compiler fails if the program does not compile. }           00397500
                                                                  00397600
program t6p6p3pld2(output);                                       00397700
type                                                              00397800
   sett     = set of 0..20;                                       00397900
   rekord   = record                                              00398000
                a : integer                                       00398100
              end;                                                00398200
   urray    = array[boolean] of boolean;                          00398300
var                                                               00398400
   setone, settwo,setthree,setfour,setfive,setsix : sett;         00398500
   recone,rectwo,recthree,recfour,recfive : rekord;               00398600
   urrayone,urraytwo,urraythree,urrayfour : urray;                00398700
                                                                  00398800
procedure testone(set1,set2,set3,set4,set5,set6 : sett;           00398900
                  rec1,rec2,rec3,rec4,rec5 : rekord;              00399000
                  urray1,urray2,urray3,urray4 : urray);           00399100
begin                                                             00399200
   write(' PASS')                                                 00399300
end;                                                              00399400
                                                                  00399500
procedure testtwo(var set1,set2,set3,set4,set5,set6 : sett;       00399600
                  var rec1,rec2,rec3,rec4,rec5 : rekord;          00399700
                  var urray1,urray2,urray3,urray4 : urray);       00399800
begin                                                             00399900
   writeln('...6.6.3.1-2')                                       00400000
end;                                                              00400100
                                                                  00400200
begin                                                             00400300
   setone:=[1];   settwo:=[1];   setthree:=[1];                   00400400
   setfour:=[1];  setfive:=[1];  setsix:=[1];                     00400500
   recone.a:=1;   rectwo.a:=1;   recthree.a:=1;                   00400600
   recfour.a:=1;  recfive.a:=1;                                   00400700
   urrayone[true]:=false;  urraytwo[true]:=false;                 00400800
   urraythree[true]:=false;   urrayfour[true]:=false;             00400900
                                                                  00401000
   testone(setone,settwo,setthree,setfour,setfive,setsix,         00401100
           recone,rectwo,recthree,recfour,recfive,                00401200
           urrayone,urraytwo,urraythree,urrayfour);               00401300
   testtwo(setone,settwo,setthree,setfour,setfive,setsix,         00401400
           recone,rectwo,recthree,recfour,recfive,                00401500
           urrayone,urraytwo,urraythree,urrayfour);               00401600
end.                                                              00401700
```

```
{TEST 6.6.3.1-3, CLASS=CONFORMANCE}                               00401800
                                                                  00401900
{ This program tests that files may be passed to                  00402000
  procedures as parameters, as a file is a type, and any          00402100
  type may be passed as a parameter.                              00402200
  The compiler fails if the program does not compile. }           00402300
                                                                  00402400
program t6p6p3pld3(output);                                       00402500
type                                                              00402600
   fyle = text;                                                   00402700
var                                                               00402800
   elyf : fyle;                                                   00402900
procedure test(var anyfile : fyle);                               00403000
begin                                                             00403100
   rewrite(anyfile);                                              00403200
   writeln(anyfile,'THIS FILE WAS A PARAMETER');                  00403300
   writeln(' PASS...6.6.3.1-3')                                   00403400
end;                                                              00403500
begin                                                             00403600
   test(elyf)                                                     00403700
end.                                                              00403800


{TEST 6.6.3.1-4, CLASS=DEVIANCE}                                  00403900
                                                                  00404000
{ The occurrence of an identifier within an identifier list of    00404100
  a parameter group is its defining occurence as a parameter      00404200
  identifier for the formal parameter list in which it occurs     00404300
  and any corresponding procedure block or function block.        00404400
  This precludes the declaration of a local variable with the same 00404500
  name as an identifier in the formal parameter list.             00404600
  Does the compiler detect this as an error, or allow it          00404700
  to occur with some form of side effect?                         00404800
  The compiler conforms if the program does not compile. }        00404900
                                                                  00405000
program t6p6p3pld4(output);                                       00405100
var                                                               00405200
   i : integer;                                                   00405300
procedure deviates(var x : integer);                              00405400
   var x : integer;                                               00405500
begin                                                             00405600
   x:=2*x;                                                        00405700
   writeln(' DEVIATES...6.6.3.1-4 : x=',x)                        00405800
end;                                                              00405900
procedure deviates1(x : integer);                                 00406000
   var x : integer;                                               00406100
begin                                                             00406200
   x:=0;                                                          00406300
   x:=2*x;                                                        00406400
   writeln(' DEVIATES...6.6.3.1-4 : x=',x)                        00406500
end;                                                              00406600
                                                                  00406700
begin                                                             00406800
   i:=5;                                                          00406900
   deviates(i);                                                   00407000
   i:=5;                                                          00407100
   deviates1(i)                                                   00407200
end.                                                              00407300
```

```
{TEST 6.6.3.1-5, CLASS=CONFORMANCE}

{ When a procedure (or function) with a parameter list is
  included in the formal parameter list of another procedure
  (or function), the identifiers in the parameter list of the
  procedure parameter have defining occurences for that list
  and the corresponding block for the procedure only, and not
  for the block of the procedure to which it is passed.
  The example in this program should be passed by the compiler. }

program t6p6p3p1d5(output);
var
   i : integer;
procedure alsoconforms(x : integer);
begin
   writeln(' PASS...6.6.3.1-5')
end;

procedure conforms(procedure alsoconforms(x : integer));
   var x : boolean;
begin
   x:=true;
   alsoconforms(1)
end;

begin
   i:=2;
   conforms(alsoconforms(i))
end.


{TEST 6.6.3.2-1, CLASS=CONFORMANCE}

{ This program would have tested that the actual parameters to
  a procedure/function are assignment compatible with the type
  of the formal parameter.
  However, this test is carried out by test 6.4.6-2, and the
  user is refered to there. }
program t6p6p3p2d1;
begin
end.


{TEST 6.6.3.3-1, CLASS=CONFORMANCE}

{ This program would have tested that the actual parameters to
  a procedure/function are identical to the type of the formal
  parameters.
  This test is carried out by program 6.4.5-1, and the user is
  refered to there. }
program t6p6p3p3d1;
begin
end.
```

```
{TEST 6.6.3.3-2, CLASS=CONFORMANCE}

{ The Pascal Standard states that any operation involving the
  formal parameter is performed immediately on the actual
  parameter. Depending on how variable parameter passing is
  implemented, this test may cause some compilers to fail.
  The compiler fails if the program does not compile, or the
  program states that this is so. }

program t6p6p3p3d2(output);
var
   direct : integer;
   pass  : boolean;
procedure indirection(var indirect : integer; var result : boolean);
   begin
      indirect:=2;
      if indirect<>direct then
         result:=false
      else
         result:=true
   end;
begin
   direct:=1;
   pass:=false;
   indirection(direct,pass);
   if pass then
      writeln(' PASS...6.6.3.3-2')
   else
      writeln(' FAIL...6.6.3.3-2')
end.
```

```
{TEST 6.6.3.3-3, CLASS=CONFORMANCE}

{ If the variable passed as a parameter involves the indexing
  of an array, or the dereferencing of a pointer, then these
  actions are executed before the activation of the block.
  The compiler fails if the program does not compile or the
  program states that this is so. }

program t6p6p3p3d3(output);
type
   rekord = record
               a : integer;
               link : ↑rekord;
               back : ↑rekord
            end;
var
   urray : array[1..2] of integer;
   i     : integer;
   temptr,ptr : ↑rekord;
procedure call(arrayloctn : integer;
               ptrderef : integer);
   begin
      i:=i+1;
      ptr:=ptr↑.link;
      if (urray[i-1] <> arrayloctn) or
         (ptr↑.back↑.a <> ptrderef) then
         writeln(' FAIL...6.6.3.3-3')
      else
         writeln(' PASS...6.6.3.3-3')
   end;
begin
   urray[1]:=1;
   urray[2]:=2;
   i:=1;
   new(ptr);
   ptr↑.a:=1;
   new(temptr);
   temptr↑.a:=2;
   ptr↑.link:=temptr;
   temptr↑.back:=ptr;
   call(urray[i],ptr↑.a)
end.
```

```
{TEST 6.6.3.4-1, CLASS=CONFORMANCE}

{ This program tests that procedures may be passed to other
  procedures and functions as parameters.
  The compiler fails if the program does not compile and run. }

program t6p6p3p4d1(output);
var
   i : integer;
procedure a(procedure b);
   begin
      write(' PASS');
      b
   end;
procedure c;
   begin
      write('.')
   end;
function d(procedure b) : integer;
   begin
      b;
      d:=2
   end;

begin
   a(c);
   i:=d(c);
   if i=2 then
      writeln('.6.6.3.4-1')
end.
```

```
{TEST 6.6.3.4-2, CLASS=CONFORMANCE}

{ This program tests that the environment of procedure
  parameters is as stated in the Pascal Standard.
  The compiler fails if the program does not compile, or
  the program states that this is so. }

program t6p6p3p4d2(output);
var
   globalone, globaltwo : integer;
procedure p(procedure f(procedure a,b);procedure g);
   var
      localtop : integer;
   procedure r;
      begin
         if globalone=1 then
         begin
            if (globaltwo<>2) or (localtop<>1) then
               writeln(' FAIL1...6.6.3.4-2')
         end
         else
            if globalone=2 then
            begin
               if (globaltwo<>2) or (localtop<>2) then
                  writeln(' FAIL2...6.6.3.4-2')
               else
                  writeln(' PASS...6.6.3.4-2')
            end
            else
               writeln(' FAIL3...6.6.3.4-2');
         globalone:=globalone+1;
      end;     { of r }
   begin    { of p }
      globaltwo:=globaltwo+1;
      localtop:=globaltwo;
      if globaltwo=1 then
         p(f,r)
      else
         f(g,r)
   end;     { of p }
procedure q(procedure f,g);
   begin
      f;
      g
   end;
procedure dummy;
begin
   writeln(' FAIL4...6.6.3.4-2')
end;

begin
   globalone:=1;
   globaltwo:=0;
   p(q,dummy)
end.
```

```
{TEST 6.6.3.5-1, CLASS=CONFORMANCE}

{ Similarly to 6.6.3.4-1, this program tests that functions may
  be passed to procedures and functions as parameters.
  The compiler fails if the program does not compile and run. }

program t6p6p3p5d1(output);
var
   j : integer;
procedure a(function b : integer);
   var
      i : integer;
   begin
      i:=b;
      write(' PASS')
   end;
function c : integer;
   begin
      c:=2
   end;
function d(function b : integer) : integer;
   begin
      d:=b
   end;
begin
   a(c);
   j:=d(c);
   if j=2 then
      writeln('...6.6.3.5-1')
end.


{TEST 6.6.3.5-2, CLASS=DEVIANCE}

{ This test checks functional compatibility in that function
  types are required to be identical. The compiler deviates if the
  program compiles and prints DEVIATES. }

program t6p6p3p5d2(output);
type
   natural=0..maxint;
var
   k:integer;

function actual(i:natural):natural;
begin
   actual:=i
end;

procedure p(function formal(i:natural):integer);
begin
   k:=formal(10)
end;

begin
   p(actual);
   writeln(' DEVIATES...6.6.3.5-2, FUNC TYPES NOT IDENTICAL')
end.
```

```
{TEST 6.6.3.6-1, CLASS=DEVIANCE}

{ This test checks that constants are not permitted
  as var parameters. The compiler deviates if the program
  compiles and prints DEVIATES. }

program t6p6p3p6d1(output);
const
   x=1;
var
   y:integer;
procedure assign(var p:integer);
begin
   p:=100
end;

begin
   assign(y);
   assign(x);   {disallowed}
   writeln(' DEVIATES...6.6.3.6-1, VAR PARAMS')
end.


{TEST 6.6.3.6-2, CLASS=DEVIANCE}

{ This test checks that parameter list compatibility is correctly
  implemented. The compiler deviates if the program compiles
  and prints DEVIATES. }

program t6p6p3p6d2(output);
type
   natural = 0..maxint;

procedure actual(i:integer; n:natural);
begin
   i:=n
end;

procedure p(procedure formal(a:integer;b:integer));
var
   k,l:integer;
begin
   k:=1; l:=2;
   formal(k,l)
end;

begin
   p(actual);
   writeln(' DEVIATES...6.6.3.6-2, VALUE PARS NOT IDENT TYPES')
end.
```

```
{TEST 6.6.3.6-3, CLASS=DEVIANCE}

{ This test checks that parameter list compatibility is correctly
  implemented. The compiler deviates if the program compiles
  and prints DEVIATES. }

program t6p6p3p6d3(output);
type
   natural = 0..maxint;

procedure actual(i:integer; n:natural);
begin
   i:=n
end;

procedure p(procedure formal(var a:integer;b:natural));
var
   k,l:integer;
begin
   k:=1; l:=2;
   formal(k,l)
end;

begin
   p(actual);
   writeln(' DEVIATES...6.6.3.6-3, VALUE/VAR MISMATCH')
end.


{TEST 6.6.3.6-4, CLASS=DEVIANCE}

{ This test checks that parameter list compatibility is correctly
  implemented. The compiler deviates if the program compiles
  and prints DEVIATES. }

program t6p6p3p6d4(output);
type
   natural = 0..maxint;

procedure actual(var i:integer;var  n:natural);
begin
   i:=n
end;

procedure p(procedure formal(var a:integer;var b:integer));
var
   k,l:integer;
begin
   k:=1; l:=2;
   formal(k,l)
end;

begin
   p(actual);
   writeln(' DEVIATES...6.6.3.6-4, VAR PARS NOT IDENT TYPES')
end.
```

```
{TEST 6.6.3.6-5, CLASS=DEVIANCE}

{ This test checks that parameter list compatibility is correctly
  implemented. The compiler deviates if the program compiles
  and prints DEVIATES. }

program t6p6p3p6d5(output);
type
   natural = 0..maxint;

procedure actual(i:integer; j:integer; n:natural);
begin
   i:=n
end;

procedure p(procedure formal(a:integer;b:integer));
var
   k,l:integer;
begin
   k:=1; l:=2;
   formal(k,l)
end;

begin
   p(actual);
   writeln(' DEVIATES...6.6.3.6-5, NO OF PARS DIFFERENT')
end.


{TEST 6.6.4.1-1, CLASS=CONFORMANCE}

{ This program tests that predefined standard procedures may
  be redefined with no conflict.
  The compiler fails if the program does not compile and run. }

program t6p6p4p1d1(output);
var
   i : integer;
procedure write(var a : integer);
   begin
      a:=a+2
   end;
procedure get(var a : integer);
   begin
      a:=a*2
   end;

begin
   i:=0;
   write(i);
   get(i);
   if i=4 then
      writeln(' PASS...6.6.4.1-1')
   else
      writeln(' FAIL...6.6.4.1-1')
end.
```

```
{TEST 6.6.5.2-1, CLASS=ERRORHANDLING}

{ This program causes an error to occur, as eof(f) does
  not yield true prior to execution of a put on the file f.
  The error should be detected at compile-time or run-time. }

program t6p6p5p2d1(output);
var
   fyle : text;
begin
   rewrite(fyle);
   writeln(fyle,'ABC');
   reset(fyle);       { eof is false and f↑='A' }
   put(fyle);         { causes an error }
   writeln(' ERROR NOT DETECTED...6.6.5.2-1')
end.


{TEST 6.6.5.2-2, CLASS=ERRORHANDLING}

{ This program causes an error to occur as eof(f) does
  not yield false prior to execution of a get on the file f.
  The error should be detected at compile-time or run-time. }

program t6p6p5p2d2(output);
var
   fyle : text;
begin
   rewrite(fyle);
   writeln(fyle,'ABC');
   reset(fyle);
   get(fyle);             { fyle↑='A' }
   get(fyle);             { fyle↑='B' }
   get(fyle);             { fyle↑='C' }
   get(fyle);             { fyle↑ undefined...eof is true }
   get(fyle);             { error since eof is true }
   writeln(' ERROR NOT DETECTED...6.6.5.2-2')
end.


{TEST 6.6.5.2-3, CLASS=CONFORMANCE}

{ This program tests if true is assigned to eof if the file f
  is empty when reset. }

program t6p6p5p2d3(output);
var
   fyle : text;
begin
   reset(fyle);
   if eof(fyle) then
      writeln(' PASS...6.6.5.2-3')
   else
      writeln(' FAIL...6.6.5.2-3')
end.
```

```
{TEST 6.6.5.2-4, CLASS=CONFORMANCE}

{ This program tests that the first element of a file f
  is assigned to the buffer variable f↑ when the procedure
  reset is used with the file f. }

program t6p6p5p2d4(output);
var
   fyle : text;
begin
   rewrite(fyle);
   writeln(fyle,'ABC');
   writeln(fyle,'DEF');
   reset(fyle);
   if fyle↑='A' then
      writeln(' PASS...6.6.5.2-4')
   else
      writeln(' FAIL...6.6.5.2-4')
end.


{TEST 6.6.5.2-5, CLASS=CONFORMANCE}

{ This program checks that a rewrite on the file f sets
  eof to be true. }

program t6p6p5p2d5(output);
var
   fyle : text;
begin
   rewrite(fyle);
   if eof(fyle) then
      writeln(' PASS...6.6.5.2-5')
   else
      writeln(' FAIL...6.6.5.2-5')
end.


{TEST 6.6.5.2-6, CLASS=ERRORHANDLING}

{ This program causes an error to occur by changing the
  current file position of a file f, while the buffer
  variable is an actual parameter to a procedure.
  The error should be detected by the compiler, or at
  run-time. }

program t6p6p5p2d6(output);
var
   fyle : text;
procedure naughty(f : char);
   begin
      if f='G' then
         put(fyle)
   end;
begin
   rewrite(fyle);
   fyle↑:='G';
   naughty(fyle↑);
   writeln(' ERROR NOT DETECTED...6.6.5.2-6')
end.
```

```
{TEST 6.6.5.2-7, CLASS=ERRORHANDLING}

{ This test is similar to 6.6.5.2-6, except that the
  buffer variable is an element of the record variable list
  of a with statement.
  The error should be detected by the compiler or at
  run-time. }

program t6p6p5p2d7(output);
type
   sex   = (male,female,notgiven);
   socialsecuritynumber = 0..10000;
   rekord = record
               a : socialsecuritynumber;
               b : sex
            end;
var
   fyle : file of rekord;
begin
   rewrite(fyle);
   with fyle↑ do
   begin
      a:=9999;
      b:=notgiven;
      put(fyle)
   end;
   writeln(' ERROR NOT DETECTED...6.6.5.2-7')
end.
```

```
{TEST 6.6.5.3-1, CLASS=CONFORMANCE}

{ This program checks that the procedure new has
  been implemented. Both forms of new are tested
  and both should pass. }

program t6p6p5p3d1(output);
type
   two      = (a,b);
   recone   = record
                i : integer;
                j : boolean
              end;
   rectwo   = record
                c : integer;
                case tagfield : two of
                   a : (m : integer);
                   b : (n : boolean)
              end;
   recthree = record
                c : integer;
                case tagfield : two of
                   a : (case tagfeeld : two of
                            a : (o : real);
                            b : (p : char));
                   b : (q : integer)
              end;
var
   ptrone : ↑recone;
   ptrtwo : ↑rectwo;
   ptrthree : ↑recthree;
begin
   new(ptrone);
   new(ptrtwo,a);
   ptrtwo↑.tagfield:=a;
   new(ptrthree,a,b);
   ptrthree↑.tagfield:=a;
   ptrthree↑.tagfeeld:=a;
   writeln(' PASS...6.6.5.3-1')
END.
```

```
{TEST 6.6.5.3-2, CLASS=CONFORMANCE}

{ This program tests that new and dispose operate as described
  in the Standard, however the undefinition of the pointer
  variable by dispose is not tested.
  The compiler fails if the program does not compile
  and run to completion. }

program t6p6p5p3d2(output);
var
   ptr : ↑integer;
   i   : integer;
begin
   for i:=1 to 10 do
   begin
      new(ptr);
      ptr↑:=i;
      dispose(ptr)
   end;
   writeln(' PASS...6.6.5.3-2')
end.
```

```
{TEST 6.6.5.3-3, CLASS=ERRORHANDLING}

{ This program causes an error to occur as the pointer
  parameter of dispose is nil. The error should be detected
  by the compiler or at run-time. }

program t6p6p5p3d3(output);
type
   rekord = record
              a : integer;
              b : boolean
            end;
var
   ptr : ↑rekord;
begin
   ptr:=nil;
   dispose(ptr);
   writeln(' ERROR NOT DETECTED...6.6.5.3-3')
end.
```

```
{TEST 6.6.5.3-4, CLASS=ERRORHANDLING}

{ Similarly to 6.6.5.3-3, an error is caused by the
  pointer variable of dispose being [illegible]
  The error should be d[illegible] compiler
  or at run[illegible]

  program t6p6p5p3d4(output);
type
   rekord = record
              a : integer;
              b : boolean
            end;
var
   ptr : ↑rekord;
begin
   dispose(ptr);
   writeln(' ERROR NOT DETECTED...6.6.5.3-4')
end.


{TEST 6.6.5.3-5, CLASS=ERRORHANDLING}

{ This program causes an error to occur as a variable
  which is currently an actual variable parameter is
  refered to by the pointer parameter of dispose.
  The error should be detected by the compiler or
  at run-time. }

program t6p6p5p3d5(output);
var
   ptr : ↑integer;
procedure error(a:integer);
   var
      x : integer;
   begin
      x:=a*2;
      dispose(ptr)
   end;
begin
   new(ptr);
   ptr↑:=6;
   error(ptr↑);
   writeln(' ERROR NOT DETECTED...6.6.5.3-5')
end.
```

```
{TEST 6.6.5.3-6, CLASS=ERRORHANDLING}

{ This program causes an error to occur as a variable which
  is an element of the record-variable-list of a with
  statement is refered to by the pointer parameter of
  dispose. }

program t6p6p5p3d6(output);
type
   subrange = 0..9999;
   rekord   = record
                name : packed array[1..15] of char;
                employeeno : subrange
              end;
var
   ptr : ↑rekord;
begin
   new(ptr);
   with ptr↑ do
   begin
      name:='HARRY M. MULLER';
      employeeno:=9998;
      dispose(ptr)
   end;
   writeln(' ERROR NOT DETECTED...6.6.5.3-6')
end.


{TEST 6.6.5.3-7, CLASS=ERRORHANDLING}

{ This program causes an error to occur, as a variable
  created by the use of the variant form of new is used
  as an operand in an expression.
  The error should be detected by the compiler, or at
  run-time. }

program t6p6p5p3d7(output);
type
   two      = (a,b);
   rekord   = record
                case tagfield:two of
                   a : (m : boolean);
                   b : (n : char)
              end;
var
   ptr      : ↑rekord;
   r        : rekord;
begin
   new(ptr,a);
   ptr↑.m:=true;
   r:=ptr↑;
   writeln(' ERROR NOT DETECTED...6.6.5.3-7')
end.
```

```
{TEST 6.6.5.3-8, CLASS=ERRORHANDLING}                                   00480100
                                                                        00480200
{ This test is similar to 6.6.5.3-7, except that the                    00480300
  variable created is used as the variable in an assignment             00480400
  statement.                                                            00480500
  The error should be detected by the compiler or at                    00480600
  run-time. }                                                           00480700
                                                                        00480800
program t6p6p5p3d8(output);                                             00480900
type                                                                    00481000
   two      = (a,b);                                                    00481100
   rekord   = record                                                    00481200
               case tagfield:two of                                     00481300
                  a : (m : boolean);                                    00481400
                  b : (n : char)                                        00481500
              end;                                                      00481600
var                                                                     00481700
   ptr : ↑rekord;                                                       00481800
   r   : rekord;                                                        00481900
begin                                                                   00482000
   new(ptr,b);                                                          00482100
   r.tagfield:=b;                                                       00482200
   r.n:='A';                                                            00482300
   ptr↑:=r;                                                             00482400
   writeln(' ERROR NOT DETECTED...6.6.5.3-8')                           00482500
end.                                                                    00482600


{TEST 6.6.5.3-9, CLASS=ERRORHANDLING}                                   00482700
                                                                        00482800
{ This test is similar to 6.6.5.3-7, except that the                    00482900
  variable created is used as an actual parameter.                      00483000
  The error should be detected by the compiler or at                    00483100
  run-time. }                                                           00483200
                                                                        00483300
program t6p6p5p3d9(output);                                             00483400
type                                                                    00483500
   two      = (a,b);                                                    00483600
   rekord   = record                                                    00483700
               case tagfield:two of                                     00483800
                  a : (m : boolean);                                    00483900
                  b : (n : char)                                        00484000
              end;                                                      00484100
var                                                                     00484200
   ptr : ↑rekord;                                                       00484300
procedure error(c : rekord);                                            00484400
   begin                                                                00484500
      writeln(' ERROR NOT DETECTED...6.6.5.3-9')                        00484600
   end;                                                                 00484700
begin                                                                   00484800
   new(ptr,a);                                                          00484900
   ptr↑.m:=true;                                                        00485000
   error(ptr↑)                                                          00485100
end.                                                                    00485200
```

```
{TEST 6.6.5.4-1, CLASS=CONFORMANCE}                                     00485300
                                                                        00485400
{ This program tests that pack and unpack are                           00485500
  implemented in this compiler as according to the                      00485600
  Standard.                                                             00485700
  The compiler fails if the program does not compile. }                 00485800
                                                                        00485900
program t6p6p5p4d1(output);                                             00486000
type                                                                    00486100
   colourtype = (red,pink,orange,yellow,green,blue);                    00486200
var                                                                     00486300
   unone    : array[3..24] of char;                                     00486400
   pacone   : packed array[1..4] of char;                               00486500
   untwo    : array[4..8] of colourtype;                                00486600
   pactwo   : packed array[6..7] of colourtype;                         00486700
   i        : integer;                                                  00486800
   colour   : colourtype;                                               00486900
begin                                                                   00487000
   pacone:='ABCD';                                                      00487100
   unpack(pacone,unone,5);                                              00487200
   colour:=red;                                                         00487300
   for i:=4 to 8 do                                                     00487400
   begin                                                                00487500
      untwo[i]:=colour;                                                 00487600
      colour:=succ(colour)                                              00487700
   end;                                                                 00487800
   pack(untwo,5,pactwo);                                                00487900
   if unone[5]='A' then                                                 00488000
      writeln(' PASS...6.6.5.4-1')                                      00488100
   else                                                                 00488200
      writeln(' FAIL...6.6.5.4-1')                                      00488300
end.                                                                    00488400
```

```
{TEST 6.6.6.1-1, CLASS=IMPLEMENTATIONDEPENDENT}

{ The Pascal Standard does not state what action takes place
  when a standard function is used as a functional
  parameter. The effect is implementation dependent.
  This program uses a standard function as a parameter to a
  procedure. The compiler may reject this as an error, or
  may permit it as it should other functional parameters. }

program t6p6p6p1d1(output);
procedure quidnunk(function a(b : integer):boolean);
   var
      x : integer;
   y : boolean;
   begin
      x:=5;
      y:=a(x);
      if x=1 then
         writeln(' STANDARD FUNCTIONS PERMITTED AS PARAMETERS',
                 '...6.6.6.1-1')
      else
         writeln(' STANDARD FUNCTIONS NOT PERMITTED AS ',
                 'PARAMETERS...6.6.6.1-1')
   end;
begin
   quidnunk(odd)
end.
```

```
{TEST 6.6.6.2-1, CLASS=CONFORMANCE}

{ This program tests the implementation of the arithmetic
  function abs. Both real and integer expressions are used.
  The compiler fails if the program does not compile and run. }

program t6p6p6p2d1(output);
const
   pi = 3.1415926;
var
   i, counter : integer;
   r : real;
function myabs1(i : integer):integer;
   begin
      if i<0 then
         myabs1:=-i
      else
         myabs1:=i
   end;
function myabs2(r:real):real;
   begin
      if r<0 then
         myabs2:=-r
      else
         myabs2:=r
   end;
begin
   counter:=0;
   for i:=-10 to 10 do
   begin
      if abs(i)=myabs1(i) then
         counter:=counter+1
   end;

   r:=-10.3;
   while r<10.3 do
   begin
      if abs(r)=myabs2(r) then
         counter:=counter+1;
      r:=r+0.9
   end;

   if counter=44 then
      writeln(' PASS...6.6.6.2-1')
   else
      writeln(' FAIL...6.6.6.2-1:ABS')
end.
```

```
{TEST 6.6.6.2-2, CLASS=CONFORMANCE}                                    00495900
                                                                       00496000
{ This program tests the implementation of the arithmetic              00496100
  function sqr. Both real and integer expressions are used.            00496200
  The compiler fails if the program does not compile and run. }        00496300
                                                                       00496400
program t6p6p6p2d2(output);                                            00496500
var                                                                    00496600
   i,counter : integer;                                                00496700
   variable : real;                                                    00496800
begin                                                                  00496900
   counter := 0;                                                       00497000
   for i:= -10 to 10 do                                                00497100
   begin                                                               00497200
      if sqr(i) = i*i then                                             00497300
         counter := counter + 1;                                       00497400
   end;                                                                00497500
   variable := -10.3;                                                  00497600
   while (variable < 10.3) do                                          00497700
   begin                                                               00497800
      if (sqr(variable) = variable*variable) then                      00497900
         counter := counter+1;                                         00498000
      variable := variable + 0.9;                                      00498100
   end;                                                                00498200
   if (counter = 44) then                                              00498300
      writeln(' PASS...6.6.6.2-2')                                     00498400
   else                                                                00498500
      writeln(' FAIL...6.6.6.2-2:SQR')                                 00498600
end.                                                                   00498700
```

```
{TEST 6.6.6.2-3, CLASS=CONFORMANCE}                                    00498800
                                                                       00498900
{ This program tests the implementation of the arithmetic              00499000
  functions sin, cos, exp, ln, sqrt, and arctan.                       00499100
  A rough accuracy test is done, but is not the purpose                00499200
  of this program.                                                     00499300
  The compiler fails if the program does not compile and run. }        00499400
                                                                       00499500
program t6p6p6p2d3(output);                                            00499600
const                                                                  00499700
   pi = 3.1415926;                                                     00499800
var                                                                    00499900
   counter : integer;                                                  00500000
begin                                                                  00500100
   counter:=0;                                                         00500200
   if (sin(pi)<0.000001) and                                           00500300
      ((0.70710<sin(pi/4)) and (sin(pi/4)<0.70711)) then               00500400
      counter:=counter+1                                               00500500
   else                                                                00500600
      writeln(' FAIL...6.6.6.2-3 : SIN');                              00500700
                                                                       00500800
   if (cos(pi)<-0.99999) and                                           00500900
      ((0.70710<cos(pi/4)) and (cos(pi/4)<0.70711)) then               00501000
      counter:=counter+1                                               00501100
   else                                                                00501200
      writeln(' FAIL...6.6.6.2-3 : COS');                              00501300
                                                                       00501400
   if ((2.71828<exp(1)) and (exp(1)<2.71829)) and                      00501500
      ((0.36787<exp(-1)) and (exp(-1)<0.36788)) and                    00501600
      ((8103.08392<exp(9)) and (exp(9)<8103.08393)) then               00501700
      counter:=counter+1                                               00501800
   else                                                                00501900
      writeln(' FAIL...6.6.6.2-3 : EXP');                              00502000
                                                                       00502100
   if (ln(exp(1))>0.99999) and                                         00502200
      ((0.69314<ln(2)) and (ln(2)<0.69315)) then                       00502300
      counter:=counter+1                                               00502400
   else                                                                00502500
      writeln(' FAIL...6.6.6.2-3 : LN');                               00502600
                                                                       00502700
   if (sqrt(25)=5) and                                                 00502800
      ((5.09901<sqrt(26)) and (sqrt(26)<5.09902)) then                 00502900
      counter:=counter+1                                               00503000
   else                                                                00503100
      writeln(' FAIL...6.6.6.2-3 : SQRT');                             00503200
                                                                       00503300
   if ((0.09966<arctan(0.1)) and (arctan(0.1)<0.09967)) and            00503400
      (arctan(0)=0) then                                               00503500
      counter:=counter+1                                               00503600
   else                                                                00503700
      writeln(' FAIL...6.6.6.2-3 : ARCTAN');                           00503800
                                                                       00503900
   if counter=6 then                                                   00504000
      writeln(' PASS...6.6.6.2-3')                                     00504100
end.                                                                   00504200
```

```
{TEST 6.6.6.2-4, CLASS=ERRORHANDLING}

{ This program causes an error to occur as an expression
  with a negative value is used as an argument for the
  arithmetic function ln.
  The error should be detected at run-time. }

program t6p6p6p2d4(output);
var
   m : real;
begin
   m:=-2.71828;
   m:=ln(m*2);
   writeln(' ERROR NOT DETECTED...6.6.6.2-4')
end.

{TEST 6.6.6.2-5, CLASS=ERRORHANDLING}

{ This program causes an error to occur as a negative
  argument is used for the sqrt function.
  The error should be detected at run-time. }

program t6p6p6p2d5(output);
var
   m    : real;
   i, j : integer;
begin
   i:=256;
   j:=i*2;
   j:=j-257;
   m:=sqrt(j-i);
   writeln(' ERROR NOT DETECTED...6.6.6.2-5')
end.
```

```
{TEST 6.6.6.2-6, CLASS=QUALITY}
{ This test checks the implementation of the sqrt function. }

program t6p6p6p2d6(output);

var


{     data required

        none

      other subprograms in this package

        machar - An environmental inquiry program providing
                 information on the floating-point arithmetic
                 system.  Note that the call to machar can
                 be deleted provided the following five
                 parameters are assigned the values indicated

                 ibeta - the radix of the floating-point system
                 it    the number of the base-ibeta digits in the
                       significand of a floating-point number
                 eps   the smallest positive floating-point
                       number such that 1.0+eps <> 1.0
                 xmin  - the smallest positive floating-point
                       number
                 xmax  - the largest finite floating-point no.

       randl(x) - A function subprogram returning logarithmically
                 distributed random real numbers.  In particular,
                       a * randl(ln(b/a))
                 is logarithmically distributed over (a,b)

        random - A function subprogram returning random real
                 numbers uniformly distributed over (0,1)


      standard subprograms required

         abs, ln, exp, sqrt                                    }

   i , ibeta , iexp ,  irnd , it , j , k , k1 , machep , maxexp ,
   iy,minexp , n , negep , ngrd : integer ;
   a , albeta , b , beta , c , eps , epsneg , r5 , r6 , r7 , sqbeta , w
   , x , xmax , xmin , xn , x1 , y , z : real ;
procedure machar (var ibeta , it , irnd , ngrd , machep , negep , iexp,
  minexp , maxexp : integer ; var eps , epsneg , xmin , xmax : real ) ;

var


{     This subroutine is intended to determine the characteristics
      of the floating-point arithmetic system that are specified
      below.  The first three are determined according to an
      algorithm due to M. Malcolm, CACM 15 (1972), pp. 949-951,
      incorporating some, but not all, of the improvements
      suggested by M. Gentleman and S. Marovich, CACM 17 (1974),
      pp. 276-277.  The version given here is for single precision.
```

```
                                                                    00513600
   Latest revision - October 1, 1976.                               00513700
                                                                    00513800
   Author - W. J. Cody                                              00513900
            Argonne National Laboratory                             00514000
                                                                    00514100
   Revised for Pascal - R. A. Freak                                 00514200
                        University of Tasmania                      00514300
                        Hobart                                      00514400
                        Tasmania                                    00514500
                                                                    00514600
   ibeta    is the radix of the floating-point representation       00514700
   it       is the number of base ibeta digits in the floating-point 00514800
              significand                                           00514900
   irnd     =  0 if the arithmetic chops,                           00515000
               1 if the arithmetic rounds                           00515100
   ngrd     =  0 if  irnd=1, or if  irnd=0  and only  it  base ibeta 00515200
               digits participate in the post normalization shift   00515300
               of the floating-point significand in multiplication  00515400
               1 if  irnd=0  and more than  it  base  ibeta digits  00515500
               participate in the post normalization shift of the   00515600
               floating-point significand in multiplication         00515700
   machep   is the exponent on the smallest positive floating-point 00515800
              number  eps such that  1.0+eps <> 1.0                 00515900
   negeps   is the exponent on the smallest positive fl. pt. no.    00516000
              negeps such that  1.0-negeps <> 1.0, except that      00516100
              negeps is bounded below by  it-3                      00516200
   iexp     is the number of bits (decimal places if ibeta = 10)    00516300
              reserved for the representation of the exponent of    00516400
              a floating-point number                               00516500
   minexp   is the exponent of the smallest positive fl. pt. no.    00516600
              xmin                                                  00516700
   maxexp   is the exponent of the largest finite floating-point    00516800
              number  xmax                                          00516900
   eps      is the smallest positive floating-point number such     00517000
              that  1.0+eps <> 1.0. in particular,                  00517100
              eps = ibeta**machep                                   00517200
   epsneg   is the smallest positive floating-point number such     00517300
              that  1.0-eps <> 1.0  (except that the exponent       00517400
              negeps is bounded below by it-3).  in particular      00517500
              epsneg = ibeta**negep                                 00517600
   xmin     is the smallest positive floating-point number.  in     00517700
              particular,  xmin = ibeta ** minexp                   00517800
   xmax     is the largest finite floating-point number.  in        00517900
              particular   xmax = (1.0-epsneg) * ibeta ** maxexp    00518000
              note - on some machines  xmax  will be only the       00518100
              second, or perhaps third, largest number, being       00518200
              too small by 1 or 2 units in the last digit of        00518300
              the significand.                                      00518400
                                                                    00518500
                                                              }     00518600
                                                                    00518700
   i , iz , j , k , mx : integer ;                                  00518800
   a , b , beta , betain , betaml , one , y , z , zero : real ;     00518900
   underflo : boolean;                                              00519000
                                                                    00519100
begin                                                               00519200
   irnd := 1 ;                                                      00519300
   one := ( irnd );                                                 00519400
   a := one + one ;                                                 00519500
   b := a ;                                                         00519600
   zero := 0.0 ;                                                    00519700
{                                                                   00519800
   determine ibeta,beta ala Malcolm                                 00519900
                                                              }     00520000
   while ( ( ( a + one ) - a ) - one = zero ) do begin              00520100
      a := a + a ;                                                  00520200
   end ;                                                            00520300
   while ( ( a + b ) - a = zero ) do begin                          00520400
      b := b + b ;                                                  00520500
   end ;                                                            00520600
   ibeta := trunc ( ( a + b ) - a );                                00520700
   beta := ( ibeta );                                               00520800
   betaml := beta - one ;                                           00520900
{                                                                   00521000
   determine irnd,ngrd,it                                           00521100
                                                              }     00521200
   if ( ( a + betaml ) - a = zero ) then irnd := 0 ;                00521300
   it := 0 ;                                                        00521400
   a := one ;                                                       00521500
   repeat begin                                                     00521600
      it := it + 1 ;                                                00521700
      a := a * beta ;                                               00521800
   end until ( ( ( a + one ) - a ) - one <> zero ) ;                00521900
{                                                                   00522000
   determine negep, epsneg                                          00522100
                                                              }     00522200
   negep := it + 3 ;                                                00522300
   a := one ;                                                       00522400
                                                                    00522500
   for i := 1 to negep do begin                                     00522600
      a := a / beta ;                                               00522700
   end ;                                                            00522800
                                                                    00522900
   while ( ( one - a ) - one = zero ) do begin                      00523000
      a := a * beta ;                                               00523100
      negep := negep - 1 ;                                          00523200
   end ;                                                            00523300
   negep := - negep ;                                               00523400
   epsneg := a ;                                                    00523500
{                                                                   00523600
   determine machep, eps                                            00523700
                                                              }     00523800
   machep := negep ;                                                00523900
   while ( ( one + a ) - one = zero ) do begin                      00524000
      a := a * beta ;                                               00524100
      machep := machep + 1 ;                                        00524200
   end ;                                                            00524300
   eps := a ;                                                       00524400
{                                                                   00524500
   determine ngrd                                                   00524600
                                                              }     00524700
   ngrd := 0 ;                                                      00524800
   if(( irnd = 0) and((( one + eps) * one - one) <> zero)) then     00524900
   ngrd := 1 ;                                                      00525000
{                                                                   00525100
   determine iexp, minexp, xmin                                     00525200
                                                                    00525300
   loop to determine largest i such that                            00525400
       (1/beta) ** (2**(i))                                         00525500
   does not underflow                                               00525600
   exit from loop is signall by an underflow                        00525700
```

```
   i := 0 ;                                                                  00525800
   betain := one / beta ;                                                    00525900
   z := betain ;                                                             00526000
   underflo := false;                                                        00526100
   repeat begin                                                              00526200
      y := z ;                                                               00526300
      z := y * y ;                                                           00526400
{                                                                            00526500
      check for underflow                                                    00526600
                                                                             00526700
      if ( ( z * one = zero ) or ( abs ( z )> y ) ) then begin               00526800
         underflo := true;                                                   00526900
      end else begin                                                         00527000
         i := i + 1 ;                                                        00527100
      end;                                                                   00527200
   end until underflo ;                                                      00527300
   k := 1 ;                                                                  00527400
{                                                                            00527500
      determine k such that (1/beta)**k does not underflow                   00527600
                                                                             00527700
      first set k = 2 ** i                                                   00527800
                                                                             00527900
                                                                             00528000
   for j := 1 to i do begin                                                  00528100
      k := k + k ;                                                           00528200
   end ;                                                                     00528300
                                                                             00528400
   iexp := i + 1 ;                                                           00528500
   mx := k + k ;                                                             00528600
   if ( ibeta = 10 ) then begin                                              00528700
{                                                                            00528800
      for decimal machines only                                              00528900
      iexp := 2 ;                                                            00529000
      iz := ibeta ;                                                          00529100
      while ( k >= iz ) do begin                                             00529200
         iz := iz * ibeta ;                                                  00529300
         iexp := iexp + 1 ;                                                  00529400
      end ;                                                                  00529500
      mx := iz + iz - 1 ;                                                    00529600
   end;                                                                      00529700
   underflo := false;                                                        00529800
   repeat begin                                                              00529900
{                                                                            00530000
      loop to construct xmin                                                 00530100
      exit from loop is signalled by an underflow                            00530200
                                                                             00530300
      xmin := y ;                                                            00530400
      y := y * betain ;                                                      00530500
      if ( ( ( y * one ) = zero ) or ( abs ( y )> xmin ) )                   00530600
      then begin                                                             00530700
         underflo := true;                                                   00530800
      end else begin                                                         00530900
         k := k + 1 ;                                                        00531000
      end;                                                                   00531100
   end until underflo ;                                                      00531200
   minexp := - k ;                                                           00531300
{  determine maxexp, xmax                                                    00531400
                                                                             00531500
   if ( ( mx <= k + k - 3 ) and ( ibeta <> 10 ) ) then begin                 00531600
      mx := mx + mx ;                                                        00531700
                                                                             00531800
      iexp := iexp + 1 ;                                                     00531900
   end;                                                                      00532000
   maxexp := mx + minexp ;                                                   00532100
{  adjust for machines with implicit leading                                 00532200
   bit in binary significand and machines with                               00532300
   radix point at extreme right of significand                               00532400
                                                                             00532500
   i := maxexp + minexp ;                                                    00532600
   if ( ( ibeta = 2 ) and ( i = 0 ) ) then maxexp := maxexp - 1 ;            00532700
   if ( i > 20 ) then maxexp := maxexp - 3 ;                                 00532800
   xmax := one - epsneg ;                                                    00532900
   if ( xmax * one <> xmax ) then xmax := one - beta * epsneg ;              00533000
   xmax := ( xmax * betain * betain * betain ) / xmin ;                      00533100
   i := maxexp + minexp + 3 ;                                                00533200
   if  ( i > 0 ) then begin                                                  00533300
                                                                             00533400
      for j := 1 to i do begin                                               00533500
         xmax := xmax * beta ;                                               00533600
      end ;                                                                  00533700
   end;                                                                      00533800
                                                                             00533900
end;                                                                         00534000
                                                                             00534100
                                                                             00534200
function random : real ;                                                     00534300
                                                                             00534400
{     random number generator - based on algorithm 266                       00534500
       by Pike and Hill (modified by Hansson)                                00534600
       collected Alg. from CACM.                                             00534700
                                                                             00534800
      This subprogram is intended for use on computers with                  00534900
       fixed point wordlength of at least 29 bIts.  it is                    00535000
       best if the floating point significand has at most                    00535100
       29 bits. }                                                            00535200
                                                                             00535300
{     The quality of the random numbers is not important.                    00535400
      If recoding is needed for small wordlength computers,                  00535500
      even returning a constant value or zero is possible. }                 00535600
                                                                             00535700
{     The value iy is global, and is initialized in the driver }             00535800
                                                                             00535900
begin                                                                        00536000
   iy := (iy*125) mod 2796203;                                               00536100
   random := ( iy )/ 2796203.0e0 ;                                           00536200
end;                                                                         00536300
                                                                             00536400
                                                                             00536500
function randl (x : real ) : real ;                                          00536600
                                                                             00536700
{     returns pseudo random numbers logarithmically distributed              00536800
      over (1,exp(x)).  thus a*randl(ln(b/a)) is logarithmically             00536900
      distributed in (a,b).                                                  00537000
                                                                             00537100
      other subroutines required                                             00537200
                                                                             00537300
         exp(x) - the exponential routine                                    00537400
                                                                             00537500
         random - a function program returning random real                   00537600
                  numbers uniformly distributed over (0,1).                  00537700
                                                                             00537800
                                                                             00537900
```

```
                                                                    00538000
begin                                                               00538100
   randl := exp ( x * random );                                     00538200
end;                                                                00538300
                                                                    00538400
                                                                    00538500
procedure printtestrun (n:integer; lb,ub:real;                      00538600
                        big,small : integer;                        00538700
                        mean,maxerror,xmaxerror,rmserror:real);     00538800
begin                                                               00538900
   writeln(' ':5,n:4,' RANDOM ARGUMENTS WERE TESTED FROM THE INTERVAL') 00539000
      ;                                                             00539100
   writeln(' ':10,'(',lb:15,',',ub:15,')');                         00539200
   writeln;                                                         00539300
   writeln(' ':5,'THE RESULT WAS TOO LARGE',big:5,' TIMES, AND');   00539400
   writeln(' ':10,'TOO SMALL',small:5,' TIMES');                    00539500
   writeln;                                                         00539600
   if (mean <> 0.0) then begin                                      00539700
      writeln(' ':5,'MEAN RELATIVE ERROR =',mean:15,'=',            00539800
         IBETA:4,' ** ',LN(ABS(mean))/ALBETA:7:2);                  00539900
   end;                                                             00540000
   if (maxerror<> 0.0) then begin                                   00540100
      writeln(' ':5,'THE MAXIMUM RELATIVE ERROR OF',maxerror:15,'=', 00540200
         IBETA:4,' ** ',LN(ABS(maxerror))/ALBETA:7:2);              00540300
      writeln(' ':10,'OCCURRED FOR X =',xmaxerror:15);              00540400
   end;                                                             00540500
   if (rmserror <> 0.0) then begin                                  00540600
      writeln(' ':5,'ROOT-MEAN-SQUARE RELATIVE ERROR =',rmserror:15, 00540700
         '=',IBETA:4,' ** ',LN(ABS(rmserror))/ALBETA:7:2);          00540800
   end;                                                             00540900
   writeln;                                                         00541000
end;   { OF PRINT TEST RUN }                                       00541100
                                                                    00541200
begin                                                               00541300
                                                                    00541400
   machar ( ibeta , it , irnd , ngrd , machep , negep , iexp , minexp , 00541500
      maxexp , eps , epsneg , xmin , xmax );                        00541600
   beta := ( ibeta );                                               00541700
   sqbeta := sqrt ( beta );                                         00541800
   albeta := ln ( beta );                                           00541900
   a := 1.0 / sqbeta ;                                              00542000
   b := 1.0 ;                                                       00542100
   n := 2000 ;                                                      00542200
   iy := 100001;                                                    00542300
{                                                                   00542400
      random argument accuracy tests                                00542500
                                                                  } 00542600
   for j := 1 to 2 do begin                                         00542700
      c := ln ( b / a );                                            00542800
      k := 0 ;                                                      00542900
      k1 := 0 ;                                                     00543000
      x1 := 0.0 ;                                                   00543100
      r5 := 0.0 ;                                                   00543200
      r6 := 0.0 ;                                                   00543300
      r7 := 0.0 ;                                                   00543400
                                                                    00543500
      for i := 1 to n do begin                                      00543600
         x := a * randl ( c ) ;                                     00543700
         y := x * x ;                                               00543800
         z := sqrt ( y );                                           00543900
         w := ( z - x ) / x ;                                       00544000
         if ( w > 0.0 ) then k := k + 1 ;                           00544100
         if ( w < 0.0 ) then k1 := k1 + 1 ;                         00544200
         r5 := r5 + w ;                                             00544300
         w := abs ( w );                                            00544400
         if ( w > r6 ) then begin                                   00544500
            r6 := w ;                                               00544600
            x1 := x ;                                               00544700
         end;                                                       00544800
         r7 := r7 + w * w ;                                         00544900
      end ;                                                         00545000
                                                                    00545100
      xn := ( n );                                                  00545200
      r5 := r5 / xn ;                                               00545300
      r7 := sqrt ( r7 / xn );                                       00545400
      writeln(' TEST OF SQRT(X*X) - X');                            00545500
      writeln;                                                      00545600
      printtestrun(n,a,b,k,k1,r5,r6,x1,r7);                         00545700
      a := 1.0 ;                                                    00545800
      b := sqbeta ;                                                 00545900
   end ;                                                            00546000
{                                                                   00546100
      special tests                                                 00546200
                                                                  } 00546300
   writeln(' TEST OF SPECIAL ARGUMENTS');                           00546400
   writeln;                                                         00546500
   x := xmin ;                                                      00546600
   y := sqrt ( x );                                                 00546700
   writeln(' SQRT(XMIN) = SQRT(',x:15,') = ',y:15);                 00546800
   writeln;                                                         00546900
   x := 1.0 - epsneg ;                                              00547000
   y := sqrt ( x );                                                 00547100
   writeln(' SQRT(1-EPSNEG) = SQRT(1-', epsneg:15, ') = ',y:15);    00547200
   writeln;                                                         00547300
   x := 1.0 ;                                                       00547400
   y := sqrt ( x );                                                 00547500
   writeln(' SQRT(1.0) = SQRT(', x:15, ') = ', y:15);               00547600
   writeln;                                                         00547700
   x := 1.0 + eps ;                                                 00547800
   y := sqrt ( x );                                                 00547900
   writeln(' SQRT(1+EPS) = SQRT(1+',eps:15, ') = ', y:15);          00548000
   writeln;                                                         00548100
   x := xmax ;                                                      00548200
   y := sqrt ( x );                                                 00548300
   writeln(' SQRT(XMAX) = SQRT(', x:15, ') = ', y:15);              00548400
   writeln;                                                         00548500
   x := 0.0 ;                                                       00548600
   y := sqrt ( x );                                                 00548700
   writeln(' SQRT(0.0) = SQRT(',x:15,') = ', y:15);                 00548800
   writeln;                                                         00548900
{                                                                   00549000
      No tests for error conditions are made here.                  00549100
      Test 6.6.6.2-5 calls sqrt with a negative argument.           00549200
                                                                    00549300
                                                                  } 00549400
   writeln(' THIS CONCLUDES THE TESTS');                            00549500
end.                                                                00549600
```

```
{TEST 6.6.6.2-7, CLASS=QUALITY}                                          00549700
                                                                         00549800
{ This test checks the implementation of the function arctan. }         00549900
                                                                         00550000
program t6p6p6p2d7(output);                                              00550100
                                                                         00550200
var                                                                      00550300
                                                                         00550400
                                                                         00550500
{     data required                                                      00550600
                                                                         00550700
         none                                                            00550800
                                                                         00550900
      subprograms required from this package                             00551000
                                                                         00551100
         machar -  as for sqrtest                                        00551200
         random -  as for sqrtest                                        00551300
                                                                         00551400
      standard subprograms required                                      00551500
                                                                         00551600
          abs, ln, arctan,  sqrt                              }          00551700
                                                                         00551800
   i , ibeta , iexp ,  irnd , ii , it , il , j , k , kl , machep ,       00551900
   iy , maxexp , minexp , n , negep , ngrd : integer ;                   00552000
   a , albeta , b , beta , betap , del , em , eps , epsneg , expon ,     00552100
   half , ob32 , one , r5 , r6 , r7 , sum , w , x , xl , xmax , xmin ,   00552200
   xn , xsq , xl , y , z , zero , zz : real ;                            00552300
                                                                         00552400
procedure machar (var ibeta , it , irnd , ngrd , machep , negep , iexp,  00552500
  minexp , maxexp : integer ; var eps , epsneg , xmin , xmax : real ) ;  00552600
                                                                         00552700
var                                                                      00552800
                                                                         00552900
                                                                         00553000
{     This subroutine is intended to determine the characteristics       00553100
      of the floating-point arithmetic system that are specified         00553200
      below.  The first three are determined according to an             00553300
      algorithm due to M. Malcolm, CACM 15 (1972), pp. 949-951,          00553400
      incorporating some, but not all, of the improvements               00553500
      suggested by M. Gentleman and S. Marovich, CACM 17 (1974),         00553600
      pp. 276-277.  The version given here is for single precision.      00553700
                                                                         00553800
      Latest revision - October 1, 1976.                                 00553900
                                                                         00554000
      Author - W. J. Cody                                                00554100
               Argonne National Laboratory                               00554200
                                                                         00554300
      Revised for Pascal - R. A. Freak                                   00554400
                           University of Tasmania                        00554500
                           Hobart                                        00554600
                           Tasmania                                      00554700
                                                                         00554800
      ibeta    is the radix of the floating-point representation         00554900
      it       is the number of base ibeta digits in the floating-point  00555000
               significand                                               00555100
      irnd     =  0 if the arithmetic chops,                             00555200
                  1 if the arithmetic rounds                             00555300
      ngrd     =  0 if  irnd=1, or if  irnd=0  and only  it  base ibeta  00555400
                    digits participate in the post normalization shift   00555500
                    of the floating-point significand in multiplication  00555600
                  1 if  irnd=0  and more than  it  base  ibeta  digits   00555700
                    participate in the post normalization shift of the   00555800
                    floating-point significand in multiplication         00555900
      machep   is the exponent on the smallest positive floating-point   00556000
               number  eps such that  1.0+eps <> 1.0                     00556100
      negeps   is the exponent on the smallest positive fl. pt. no.      00556200
               negeps such that  1.0-negeps <> 1.0, except that          00556300
               negeps is bounded below by  it-3                          00556400
      iexp     is the number of bits (decimal places if ibeta = 10)      00556500
               reserved for the representation of the exponent of        00556600
               a floating-point number                                   00556700
      minexp   is the exponent of the smallest positive fl. pt. no.      00556800
               xmin                                                      00556900
      maxexp   is the exponent of the largest finite floating-point      00557000
               number  xmax                                              00557100
      eps      is the smallest positive floating-point number such       00557200
               that  1.0+eps <> 1.0. in particular,                      00557300
               eps = ibeta**machep                                       00557400
      epsneg   is the smallest positive floating-point number such       00557500
               that  1.0-eps <> 1.0  (except that the exponent           00557600
               negeps is bounded below by it-3).  in particular          00557700
               epsneg = ibeta**negep                                     00557800
      xmin     is the smallest positive floating-point number.  in       00557900
               particular,  xmin = ibeta ** minexp                       00558000
      xmax     is the largest finite floating-point number.  in          00558100
               particular   xmax = (1.0-epsneg) * ibeta ** maxexp        00558200
               note - on some machines  xmax  will be only the           00558300
               second, or perhaps third, largest number, being           00558400
               too small by 1 or 2 units in the last digit of            00558500
               the significand.                                          00558600
                                                                         00558700
                                                                    }    00558800
                                                                         00558900
   i , iz , j , k , mx : integer ;                                       00559000
   a , b , beta , betain , betaml , one , y , z , zero : real ;          00559100
   underflo : boolean;                                                   00559200
                                                                         00559300
begin                                                                    00559400
   irnd := 1 ;                                                           00559500
   one := ( irnd );                                                      00559600
   a := one + one ;                                                      00559700
   b := a ;                                                              00559800
   zero := 0.0 ;                                                         00559900
{                                                                        00560000
      determine ibeta,beta ala Malcolm                                   00560100
                                                                    }    00560200
   while ( ( ( a + one ) - a ) - one = zero ) do begin                   00560300
      a := a + a ;                                                       00560400
   end ;                                                                 00560500
   while ( ( a + b ) - a = zero ) do begin                               00560600
      b := b + b ;                                                       00560700
   end ;                                                                 00560800
   ibeta := trunc ( ( a + b ) - a );                                     00560900
   beta := ( ibeta );                                                    00561000
   betaml := beta - one ;                                                00561100
{                                                                        00561200
      determine irnd,ngrd,it                                             00561300
                                                                    }    00561400
   if ( ( a + betaml ) - a = zero ) then irnd := 0 ;                     00561500
   it := 0 ;                                                             00561600
   a := one ;                                                            00561700
   repeat begin                                                          00561800
```

```
      it := it + 1 ;                                                    00561900
      a := a * beta ;                                                   00562000
   end until ( ( ( a + one ) - a ) - one <> zero ) ;                    00562100
{                                                                       00562200
      determine negep, epsneg                                           00562300
}                                                                       00562400
   negep := it + 3 ;                                                    00562500
   a := one ;                                                           00562600
                                                                        00562700
   for i := 1 to negep do begin                                         00562800
      a := a / beta ;                                                   00562900
   end ;                                                                00563000
                                                                        00563100
   while ( ( one - a ) - one = zero ) do begin                          00563200
      a := a * beta ;                                                   00563300
      negep := negep - 1 ;                                              00563400
   end ;                                                                00563500
   negep := - negep ;                                                   00563600
   epsneg := a ;                                                        00563700
{                                                                       00563800
      determine machep, eps                                             00563900
}                                                                       00564000
   machep := negep ;                                                    00564100
   while ( ( one + a ) - one = zero ) do begin                          00564200
      a := a * beta ;                                                   00564300
      machep := machep + 1 ;                                            00564400
   end ;                                                                00564500
   eps := a ;                                                           00564600
{                                                                       00564700
      determine ngrd                                                    00564800
}                                                                       00564900
   ngrd := 0 ;                                                          00565000
   if(( irnd = 0) and((( one + eps) * one - one) <> zero)) then         00565100
   ngrd := 1 ;                                                          00565200
{                                                                       00565300
      determine iexp, minexp, xmin                                      00565400
                                                                        00565500
      loop to determine largest i such that                             00565600
          (1/beta) ** (2**(i))                                          00565700
      does not underflow                                                00565800
      exit from loop is signall by an underflow                         00565900
}                                                                       00566000
   i := 0 ;                                                             00566100
   betain := one / beta ;                                               00566200
   z := betain ;                                                        00566300
   underflo := false;                                                   00566400
   repeat begin                                                         00566500
      y := z ;                                                          00566600
      z := y * y ;                                                      00566700
{                                                                       00566800
      check for underflow                                               00566900
}                                                                       00567000
      if ( ( z * one = zero ) or ( abs ( z )> y ) ) then begin          00567100
         underflo := true;                                              00567200
      end else begin                                                    00567300
         i := i + 1 ;                                                   00567400
      end;                                                              00567500
   end until underflo ;                                                 00567600
   k := 1 ;                                                             00567700
{                                                                       00567800
      determine k such that (1/beta)**k does not underflow              00567900
```

```
      first set k = 2 ** i                                              00568000
}                                                                       00568100
                                                                        00568200
   for j := 1 to i do begin                                             00568300
      k := k + k ;                                                      00568400
   end ;                                                                00568500
                                                                        00568600
   iexp := i + 1 ;                                                      00568700
   mx := k + k ;                                                        00568800
   if ( ibeta = 10 ) then begin                                         00568900
{                                                                       00569000
      for decimal machines only                                         00569100
}                                                                       00569200
      iexp := 2 ;                                                       00569300
      iz := ibeta ;                                                     00569400
      while ( k >= iz ) do begin                                        00569500
         iz := iz * ibeta ;                                             00569600
         iexp := iexp + 1 ;                                             00569700
      end ;                                                             00569800
      mx := iz + iz - 1 ;                                               00569900
   end;                                                                 00570000
   underflo := false;                                                   00570100
   repeat begin                                                         00570200
{                                                                       00570300
      loop to construct xmin                                            00570400
      exit from loop is signalled by an underflow                       00570500
}                                                                       00570600
      xmin := y ;                                                       00570700
      y := y * betain ;                                                 00570800
      if ( ( ( y * one ) = zero ) or ( abs ( y )> xmin ) )              00570900
      then begin                                                        00571000
         underflo := true;                                              00571100
      end else begin                                                    00571200
         k := k + 1 ;                                                   00571300
      end;                                                              00571400
   end until underflo ;                                                 00571500
   minexp := - k ;                                                      00571600
{  determine maxexp, xmax                                               00571700
}                                                                       00571800
   if ( ( mx <= k + k - 3 ) and ( ibeta <> 10 ) ) then begin            00571900
      mx := mx + mx ;                                                   00572000
      iexp := iexp + 1 ;                                                00572100
   end;                                                                 00572200
   maxexp := mx + minexp ;                                              00572300
{  adjust for machines with implicit leading                            00572400
   bit in binary significand and machines with                          00572500
   radix point at extreme right of significand                          00572600
}                                                                       00572700
   i := maxexp + minexp ;                                               00572800
   if ( ( ibeta = 2 ) and ( i = 0 ) ) then maxexp := maxexp - 1 ;       00572900
   if ( i > 20 ) then maxexp := maxexp - 3 ;                            00573000
   xmax := one - epsneg ;                                               00573100
   if ( xmax * one <> xmax ) then xmax := one - beta * epsneg ;         00573200
   xmax := ( xmax * betain * betain * betain ) / xmin ;                 00573300
   i := maxexp + minexp + 3 ;                                           00573400
   if  ( i > 0 ) then begin                                             00573500
                                                                        00573600
      for j := 1 to i do begin                                          00573700
         xmax := xmax * beta ;                                          00573800
      end ;                                                             00573900
   end;                                                                 00574000
```

```
end;                                                                    00574100
                                                                        00574200
                                                                        00574300
function random : real ;                                                00574400
                                                                        00574500
{     random number generator - based on algorithm 266                  00574600
       by Pike and Hill (modified by Hansson)                           00574700
       collected Alg. from CACM.                                        00574800
                                                                        00574900
      This subprogram is intended for use on computers with             00575000
       fixed point wordlength of at least 29 bIts.  it is               00575100
       best if the floating point significand has at most               00575200
       29 bits. }                                                       00575300
                                                                        00575400
{     The quality of the random numbers is not important.               00575500
      If recoding is needed for small wordlength computers,             00575600
      even returning a constant value or zero is possible. }            00575700
                                                                        00575800
{     The value iy is global, and is initialized in the driver }        00575900
                                                                        00576000
begin                                                                   00576100
   iy := (iy*125) mod 2796203;                                          00576200
   random := ( iy )/ 2796203.0e0 ;                                      00576300
end;                                                                    00576400
                                                                        00576500
                                                                        00576600
procedure printtestrun (n:integer; lb,ub:real;                          00576700
                        big,small : integer;                            00576800
                        mean,maxerror,xmaxerror,rmserror:real);         00576900
begin                                                                   00577000
   writeln(' ':5,n:4,' RANDOM ARGUMENTS WERE TESTED FROM THE INTERVAL')  00577100
      ;                                                                 00577200
   writeln(' ':10,'(',lb:15,',',ub:15,')');                             00577300
   writeln;                                                             00577400
   writeln(' ':5,'THE RESULT WAS TOO LARGE',big:5,' TIMES, AND');       00577500
   writeln(' ':10,'TOO SMALL',small:5,' TIMES');                        00577600
   writeln;                                                             00577700
   if (mean <> 0.0) then begin                                          00577800
      writeln(' ':5,'MEAN RELATIVE ERROR =',mean:15,'=',                00577900
         IBETA:4,' ** ',LN(ABS(mean))/ALBETA:7:2);                      00578000
   end;                                                                 00578100
   if (maxerror<> 0.0) then begin                                       00578200
      writeln(' ':5,'THE MAXIMUM RELATIVE ERROR OF',maxerror:15,'=',    00578300
         IBETA:4,' ** ',LN(ABS(maxerror))/ALBETA:7:2);                  00578400
      writeln(' ':10,'OCCURRED FOR X =',xmaxerror:15);                  00578500
   end;                                                                 00578600
   if (rmserror <> 0.0) then begin                                      00578700
      writeln(' ':5,'ROOT-MEAN-SQUARE RELATIVE ERROR =',rmserror:15,    00578800
         '=',IBETA:4,' ** ',LN(ABS(rmserror))/ALBETA:7:2);              00578900
   end;                                                                 00579000
   writeln;                                                             00579100
end;   { OF PRINT TEST RUN }                                           00579200
                                                                        00579300
                                                                        00579400
begin                                                                   00579500
   iy := 100001;                                                        00579600
   machar ( ibeta , it , irnd , ngrd , machep , negep , iexp , minexp , 00579700
      maxexp , eps , epsneg , xmin , xmax );                            00579800
   beta := ( ibeta );                                                   00579900
   albeta := ln ( beta );                                               00580000
                                                                        00580100
   one := 1.0 ;                                                         00580200
   half := 0.5 ;                                                        00580300
   zero := 0.0 ;                                                        00580400
   a := - 0.0625 ;                                                      00580500
   b := - a ;                                                           00580600
   ob32 := b * half ;                                                   00580700
   n := 2000 ;                                                          00580800
   xn := ( n );                                                         00580900
   il := 0 ;                                                            00581000
{                                                                       00581100
      random argument accuracy tests                                    00581200
}                                                                       00581300
   for j := 1 to 4 do begin                                             00581400
      k := 0 ;                                                          00581500
      k1 := 0 ;                                                         00581600
      xl := zero ;                                                      00581700
      r5 := zero ;                                                      00581800
      r6 := zero ;                                                      00581900
      r7 := zero ;                                                      00582000
      del := ( b - a ) / xn ;                                           00582100
      xl := a ;                                                         00582200
                                                                        00582300
      for i := 1 to n do begin                                          00582400
         x := del * random + xl ;                                       00582500
         if ( j = 2 ) then x := ( ( 1.0 + x * a ) - one ) * 16.0 ;      00582600
         z := arctan ( x );                                             00582700
         case j of                                                      00582800
         1:                                                             00582900
            begin                                                       00583000
               xsq := x * x ;                                           00583100
               em := 17.0 ;                                             00583200
               sum := xsq / em ;                                        00583300
                                                                        00583400
               for ii := 1 to 7 do begin                                00583500
                  em := em - 2.0 ;                                      00583600
                  sum := ( one / em - sum ) * xsq ;                     00583700
               end ;                                                    00583800
                                                                        00583900
               zz := x - x * sum ;                                      00584000
            end;                                                        00584100
         2:                                                             00584200
            begin                                                       00584300
               y := x - 0.0625 ;                                        00584400
               y := y / ( one + x * a ) ;                               00584500
               zz := ( arctan ( y )- 8.119000404265152602le-5 ) +       00584600
                  ob32 ;                                                00584700
               zz := zz + ob32 ;                                        00584800
            end;                                                        00584900
         3,4:                                                           00585000
            begin                                                       00585100
               z := z + z ;                                             00585200
               y := x / ( ( half + x * half ) * ( ( half - x ) + half ) 00585300
                  ) ;                                                   00585400
               zz := arctan ( y );                                      00585500
            end;                                                        00585600
         end;                                                           00585700
         w := ( z - zz ) / z ;                                          00585800
         if ( w > zero ) then k := k + 1 ;                              00585900
         if ( w < zero ) then k1 := k1 + 1 ;                            00586000
         r5 := r5 + w ;                                                 00586100
         w := abs ( w );                                                00586200
```

```
      if ( w > r6 ) then begin                                          00586300
         r6 := w ;                                                      00586400
         xl := x ;                                                      00586500
      end;                                                              00586600
      r7 := r7 + w * w ;                                                00586700
      xl := xl + del ;                                                  00586800
   end ;                                                                00586900
                                                                        00587000
   r5 := r5 / xn ;                                                      00587100
   r7 := sqrt ( r7 / xn );                                              00587200
   if ( j = 1 ) then begin                                              00587300
      writeln(' TEST OF ARCTAN(X) VS TRUNCATED TAYLOR SERIES');         00587400
      writeln;                                                          00587500
   end;                                                                 00587600
   if ( j = 2 ) then begin                                              00587700
      write(' TEST OF ARCTAN(X) VS ARCTAN(1/16) + ');                   00587800
      writeln(' ARCTAN((X-1/16)/(1+X/16))');                            00587900
      writeln;                                                          00588000
   end;                                                                 00588100
   if ( j > 2 ) then begin                                              00588200
      writeln(' TEST OF 2*ARCTAN(X) VS ARCTAN(2X/(1-X*X))');            00588300
      writeln;                                                          00588400
   end;                                                                 00588500
   printtestrun(n,a,b,k,k1,r5,r6,xl,r7);                                00588600
   a := b ;                                                             00588700
   if ( j = 1 ) then b := 2.0 - sqrt ( 3.0 );                           00588800
   if ( j = 2 ) then b := sqrt ( 2.0 )- one ;                           00588900
   if ( j = 3 ) then b := one ;                                         00589000
end ;                                                                   00589100
{                                                                       00589200
   special tests                                                        00589300
                                                                      } 00589400
writeln(' THE IDENTITY ARCTAN(-X) = -ARCTAN(X) WILL BE TESTED');        00589500
writeln(' ':7, 'X', ' ':9, 'F(X) + F(-X)');                             00589600
writeln;                                                                00589700
a := 5.0 ;                                                              00589800
                                                                        00589900
for i := 1 to 5 do begin                                                00590000
   x := random * a ;                                                    00590100
   z := arctan ( x )+ arctan ( - x );                                   00590200
   writeln(x:14, z:15);                                                 00590300
end ;                                                                   00590400
writeln;                                                                00590500
                                                                        00590600
writeln(' THE IDENTITY ARCTAN(X) = X, X SMALL, WILL BE TESTED');        00590700
writeln(' ':7, 'X', ' ':9, 'X - F(X)');                                 00590800
writeln;                                                                00590900
betap := exp ( it * ln( beta ));                                        00591000
x := random / betap ;                                                   00591100
                                                                        00591200
for i := 1 to 5 do begin                                                00591300
   z := x - arctan ( x );                                               00591400
   writeln(x:14, z:15);                                                 00591500
   x := x / beta ;                                                      00591600
end ;                                                                   00591700
writeln;                                                                00591800
writeln;                                                                00591900
                                                                        00592000
writeln(' TEST OF UNDERFLOW FOR A VERY SMALL ARGUMENT');                00592100
writeln;                                                                00592200
expon := ( minexp )* 0.75 ;                                             00592300
   x := exp ( expon * ln( beta ));                                      00592400
   y := arctan ( x );                                                   00592500
   writeln(' ':5, ' ARCTAN(', x:13, ') = ', y:13);                      00592600
   writeln;                                                             00592700
   write(' THE FUNCTION WILL BE CALLED WITH THE ARGUMENT');              00592800
   writeln(xmax:15);                                                    00592900
   z := arctan ( xmax );                                                00593000
   writeln(' ':5, ' ARCTAN(', xmax:13, ') = ', z:13);                   00593100
   writeln;                                                             00593200
   writeln(' THIS CONCLUDES THE TESTS');                                00593300
end.                                                                    00593400
```

```
{TEST 6.6.6.2-8, CLASS=QUALITY}                                          00593500
                                                                         00593600
{ This test checks the implementation of the exp function. }            00593700
                                                                         00593800
program t6p6p6p2d8(output);                                              00593900
                                                                         00594000
var                                                                      00594100
                                                                         00594200
                                                                         00594300
{     data required                                                      00594400
                                                                         00594500
         none                                                            00594600
                                                                         00594700
      other subprograms in this package                                  00594800
                                                                         00594900
         machar -  as for sqrtest                                        00595000
         random -  as for sqrtest                                        00595100
                                                                         00595200
                                                                         00595300
      standard subprograms required                                      00595400
                                                                         00595500
         abs, ln, exp, sqrt                                              00595600
                                                                       } 00595700
                                                                         00595800
   i , ibeta , iexp ,  irnd , it , il , j , k , kl , machep ,            00595900
   iy , maxexp , minexp , n , negep , ngrd : integer ;                   00596000
   a , albeta , b , beta , d , del , eps , epsneg , r5 , r6 , r7 , v ,   00596100
   w , x , xl , xmax , xmin , xn , xl , y , z , zz : real ;              00596200
                                                                         00596300
procedure machar (var ibeta , it , irnd , ngrd , machep , negep , iexp,  00596400
  minexp , maxexp : integer ; var eps , epsneg , xmin , xmax : real ) ;  00596500
                                                                         00596600
var                                                                      00596700
                                                                         00596800
                                                                         00596900
{     This subroutine is intended to determine the characteristics       00597000
      of the floating-point arithmetic system that are specified         00597100
      below.  The first three are determined according to an             00597200
      algorithm due to M. Malcolm, CACM 15 (1972), pp. 949-951,          00597300
      incorporating some, but not all, of the improvements               00597400
      suggested by M. Gentleman and S. Marovich, CACM 17 (1974),         00597500
      pp. 276-277.  The version given here is for single precision.      00597600
                                                                         00597700
      Latest revision - October 1, 1976.                                 00597800
                                                                         00597900
      Author - W. J. Cody                                                00598000
               Argonne National Laboratory                               00598100
                                                                         00598200
      Revised for Pascal - R. A. Freak                                   00598300
                           University of Tasmania                        00598400
                           Hobart                                        00598500
                           Tasmania                                      00598600
                                                                         00598700
      ibeta    is the radix of the floating-point representation         00598800
      it       is the number of base ibeta digits in the floating-point  00598900
               significand                                               00599000
      irnd     =  0 if the arithmetic chops,                             00599100
                  1 if the arithmetic rounds                             00599200
      ngrd     =  0 if  irnd=1, or if  irnd=0  and only  it  base ibeta  00599300
                  digits participate in the post normalization shift     00599400
                  of the floating-point significand in multiplication    00599500
                  1 if  irnd=0  and more than  it  base  ibeta  digits   00599600
                  participate in the post normalization shift of the     00599700
                  floating-point significand in multiplication           00599800
      machep   is the exponent on the smallest positive floating-point   00599900
               number  eps such that  1.0+eps <> 1.0                     00600000
      negeps   is the exponent on the smallest positive fl. pt. no.      00600100
               negeps such that  1.0-negeps <> 1.0, except that          00600200
               negeps is bounded below by  it-3                          00600300
      iexp     is the number of bits (decimal places if ibeta = 10)      00600400
               reserved for the representation of the exponent of        00600500
               a floating-point number                                   00600600
      minexp   is the exponent of the smallest positive fl. pt. no.      00600700
               xmin                                                      00600800
      maxexp   is the exponent of the largest finite floating-point      00600900
               number  xmax                                              00601000
      eps      is the smallest positive floating-point number such       00601100
               that  1.0+eps <> 1.0. in particular,                      00601200
               eps = ibeta**machep                                       00601300
      epsneg   is the smallest positive floating-point number such       00601400
               that  1.0-eps <> 1.0  (except that the exponent           00601500
               negeps is bounded below by it-3).  in particular          00601600
               epsneg = ibeta**negep                                     00601700
      xmin     is the smallest positive floating-point number.  in       00601800
               particular,  xmin = ibeta ** minexp                       00601900
      xmax     is the largest finite floating-point number.  in          00602000
               particular   xmax = (1.0-epsneg) * ibeta ** maxexp        00602100
               note - on some machines  xmax  will be only the           00602200
               second, or perhaps third, largest number, being           00602300
               too small by 1 or 2 units in the last digit of            00602400
               the significand.                                          00602500
                                                                         00602600
                                                                       } 00602700
                                                                         00602800
   i , iz , j , k , mx : integer ;                                       00602900
   a , b , beta , betain , betaml , one , y , z , zero : real ;          00603000
   underflo : boolean;                                                   00603100
                                                                         00603200
begin                                                                    00603300
   irnd := 1 ;                                                           00603400
   one := ( irnd );                                                      00603500
   a := one + one ;                                                      00603600
   b := a ;                                                              00603700
   zero := 0.0 ;                                                         00603800
{                                                                        00603900
      determine ibeta,beta ala Malcolm                                   00604000
                                                                       } 00604100
   while ( ( ( a + one ) - a ) - one = zero ) do begin                   00604200
      a := a + a ;                                                       00604300
   end ;                                                                 00604400
   while ( ( a + b ) - a = zero ) do begin                               00604500
      b := b + b ;                                                       00604600
   end ;                                                                 00604700
   ibeta := trunc ( ( a + b ) - a );                                     00604800
   beta := ( ibeta );                                                    00604900
   betaml := beta - one ;                                                00605000
{                                                                        00605100
      determine irnd,ngrd,it                                             00605200
                                                                       } 00605300
   if ( ( a + betaml ) - a = zero ) then irnd := 0 ;                     00605400
   it := 0 ;                                                             00605500
   a := one ;                                                            00605600
```

```
   repeat begin                                                          00605700
      it := it + 1 ;                                                     00605800
      a := a * beta ;                                                    00605900
   end until ( ( ( a + one ) - a ) - one <> zero ) ;                     00606000
{                                                                        00606100
      determine negep, epsneg                                            00606200
}                                                                        00606300
                                                                         00606400
   negep := it + 3 ;                                                     00606500
   a := one ;                                                            00606600
                                                                         00606700
   for i := 1 to negep do begin                                          00606800
      a := a / beta ;                                                    00606900
   end ;                                                                 00607000
                                                                         00607100
   while ( ( one - a ) - one = zero ) do begin                           00607200
      a := a * beta ;                                                    00607300
      negep := negep - 1 ;                                               00607400
   end ;                                                                 00607500
   negep := - negep ;                                                    00607600
   epsneg := a ;                                                         00607700
{                                                                        00607800
      determine machep, eps                                              00607900
}                                                                        00608000
                                                                         00608100
   machep := negep ;                                                     00608200
   while ( ( one + a ) - one = zero ) do begin                           00608300
      a := a * beta ;                                                    00608400
      machep := machep + 1 ;                                             00608500
   end ;                                                                 00608600
   eps := a ;                                                            00608700
{                                                                        00608800
      determine ngrd                                                     00608900
}                                                                        00609000
                                                                         00609100
   ngrd := 0 ;                                                           00609200
   if(( irnd = 0) and(((one + eps) * one - one) <> zero)) then           00609300
   ngrd := 1 ;                                                           00609400
{                                                                        00609500
      determine iexp, minexp, xmin                                       00609600
                                                                         00609700
      loop to determine largest i such that                              00609800
          (1/beta) ** (2**(i))                                           00609900
      does not underflow                                                 00610000
      exit from loop is signall by an underflow                          00610100
                                                                         00610200
   i := 0 ;                                                              00610300
   betain := one / beta ;                                                00610400
   z := betain ;                                                         00610500
   underflo := false;                                                    00610600
   repeat begin                                                          00610700
      y := z ;                                                           00610800
      z := y * y ;                                                       00610900
{                                                                        00611000
      check for underflow                                                00611100
                                                                         00611200
      if ( ( z * one = zero ) or ( abs ( z )> y ) ) then begin           00611300
         underflo := true;                                               00611400
      end else begin                                                     00611500
         i := i + 1 ;                                                    00611600
      end;                                                               00611700
   end until underflo ;                                                  00611800
   k := 1 ;                                                              00611900
{                                                                        00612000
      determine k such that (1/beta)**k does not underflow               00612100
                                                                         00612200
      first set k = 2 ** i                                               00612300
}                                                                        00612400
                                                                         00612500
   for j := 1 to i do begin                                              00612600
      k := k + k ;                                                       00612700
   end ;                                                                 00612800
                                                                         00612900
   iexp := i + 1 ;                                                       00613000
   mx := k + k ;                                                         00613100
   if ( ibeta = 10 ) then begin                                          00613200
{                                                                        00613300
      for decimal machines only                                          00613400
      iexp := 2 ;                                                        00613500
      iz := ibeta ;                                                      00613600
      while ( k >= iz ) do begin                                         00613700
         iz := iz * ibeta ;                                              00613800
         iexp := iexp + 1 ;                                              00613900
      end ;                                                              00614000
      mx := iz + iz - 1 ;                                                00614100
   end;                                                                  00614200
   underflo := false;                                                    00614300
   repeat begin                                                          00614400
{                                                                        00614500
      loop to construct xmin                                             00614600
      exit from loop is signalled by an underflow                        00614700
                                                                         00614800
      xmin := y ;                                                        00614900
      y := y * betain ;                                                  00615000
      if ( ( ( y * one ) = zero ) or ( abs ( y )> xmin ) )               00615100
      then begin                                                         00615200
         underflo := true;                                               00615300
      end else begin                                                     00615400
         k := k + 1 ;                                                    00615500
      end;                                                               00615600
   end until underflo ;                                                  00615700
   minexp := - k ;                                                       00615800
{  determine maxexp, xmax                                                00615900
}                                                                        00616000
   if ( ( mx <= k + k - 3 ) and ( ibeta <> 10 ) ) then begin             00616100
      mx := mx + mx ;                                                    00616200
      iexp := iexp + 1 ;                                                 00616300
   end;                                                                  00616400
   maxexp := mx + minexp ;                                               00616500
{  adjust for machines with implicit leading                             00616600
   bit in binary significand and machines with                           00616700
   radix point at extreme right of significand                           00616800
}                                                                        00616900
   i := maxexp + minexp ;                                                00617000
   if ( ( ibeta = 2 ) and ( i = 0 ) ) then maxexp := maxexp - 1 ;        00617100
   if ( i > 20 ) then maxexp := maxexp - 3 ;                             00617200
   xmax := one - epsneg ;                                                00617300
   if ( xmax * one <> xmax ) then xmax := one - beta * epsneg ;          00617400
   xmax := ( xmax * betain * betain * betain ) / xmin ;                  00617500
   i := maxexp + minexp + 3 ;                                            00617600
   if  ( i > 0 ) then begin                                              00617700
                                                                         00617800
      for j := 1 to i do begin
         xmax := xmax * beta ;
      end ;
```

```
   end;                                                                  00617900
                                                                         00618000
end;                                                                     00618100
                                                                         00618200
                                                                         00618300
function random : real ;                                                 00618400
                                                                         00618500
{     random number generator - based on algorithm 266                   00618600
       by Pike and Hill (modified by Hansson)                            00618700
       collected Alg. from CACM.                                         00618800
                                                                         00618900
      This subprogram is intended for use on computers with              00619000
       fixed point wordlength of at least 29 bIts.  it is                00619100
       best if the floating point significand has at most                00619200
       29 bits. }                                                        00619300
                                                                         00619400
{     The quality of the random numbers is not important.                00619500
      If recoding is needed for small wordlength computers,              00619600
      even returning a constant value or zero is possible. }             00619700
                                                                         00619800
{     The value iy is global, and is initialized in the driver }         00619900
                                                                         00620000
begin                                                                    00620100
   iy := (iy*125) mod 2796203;                                           00620200
   random := ( iy )/ 2796203.0e0 ;                                       00620300
end;                                                                     00620400
                                                                         00620500
                                                                         00620600
procedure printtestrun (n:integer; lb,ub:real;                           00620700
                        big,small : integer;                             00620800
                        mean,maxerror,xmaxerror,rmserror:real);          00620900
begin                                                                    00621000
   writeln(' ':5,n:4,' RANDOM ARGUMENTS WERE TESTED FROM THE INTERVAL')  00621100
      ;                                                                  00621200
   writeln(' ':10,'(',lb:15,',',ub:15,')');                              00621300
   writeln;                                                              00621400
   writeln(' ':5,'THE RESULT WAS TOO LARGE',big:5,' TIMES, AND');        00621500
   writeln(' ':10,'TOO SMALL',small:5,' TIMES');                         00621600
   writeln;                                                              00621700
   if (mean <> 0.0) then begin                                           00621800
      writeln(' ':5,'MEAN RELATIVE ERROR =',mean:15,'=',                 00621900
         IBETA:4,' ** ',LN(ABS(mean))/ALBETA:7:2);                       00622000
   end;                                                                  00622100
   if (maxerror<> 0.0) then begin                                        00622200
      writeln(' ':5,'THE MAXIMUM RELATIVE ERROR OF',maxerror:15,'=',     00622300
         IBETA:4,' ** ',LN(ABS(maxerror))/ALBETA:7:2);                   00622400
      writeln(' ':10,'OCCURRED FOR X =',xmaxerror:15);                   00622500
   end;                                                                  00622600
   if (rmserror <> 0.0) then begin                                       00622700
      writeln(' ':5,'ROOT-MEAN-SQUARE RELATIVE ERROR =',rmserror:15,     00622800
         '=',IBETA:4,' ** ',LN(ABS(rmserror))/ALBETA:7:2);               00622900
   end;                                                                  00623000
   writeln;                                                              00623100
end;   { OF PRINT TEST RUN }                                            00623200
                                                                         00623300
                                                                         00623400
begin                                                                    00623500
   iy := 100001;                                                         00623600
   machar ( ibeta , it , irnd , ngrd , machep , negep , iexp , minexp ,  00623700
      maxexp , eps , epsneg , xmin , xmax );                             00623800
   beta := ( ibeta );                                                    00623900
   albeta := ln ( beta );                                                00624000
   v := 0.0625 ;                                                         00624100
   a := 2.0 ;                                                            00624200
   b := ln ( a )* 0.5 ;                                                  00624300
   a := - b + v ;                                                        00624400
   d := 0.9 * xmax ;                                                     00624500
   d := ln ( d );                                                        00624600
   n := 2000 ;                                                           00624700
   xn := ( n );                                                          00624800
   il := 0 ;                                                             00624900
{                                                                        00625000
      random argument accuracy tests                                     00625100
                                                                       } 00625200
   for j := 1 to 3 do begin                                              00625300
      k := 0 ;                                                           00625400
      k1 := 0 ;                                                          00625500
      x1 := 0.0 ;                                                        00625600
      r5 := 0.0 ;                                                        00625700
      r6 := 0.0 ;                                                        00625800
      r7 := 0.0 ;                                                        00625900
      del := ( b - a ) / xn ;                                            00626000
      x1 := a ;                                                          00626100
                                                                         00626200
      for i := 1 to n do begin                                           00626300
         x := del * random + x1 ;                                        00626400
         y := x - v ;                                                    00626500
         if ( y < 0.0 ) then x := y + v ;                                00626600
         z := exp ( x );                                                 00626700
         zz := exp ( y );                                                00626800
         if ( j = 1 ) then begin                                         00626900
            z := z - z * 6.058693718652421388e-2 ;                       00627000
         end else begin                                                  00627100
            if ( ibeta = 10 ) then z := z * 6.0e-2 + z *                 00627200
               5.466789530794296106e-5                                   00627300
            else z := z * 0.0625 - z *                                   00627400
               2.4453321046920570389e-3 ;                                00627500
         end;                                                            00627600
         w := ( z - zz ) / zz ;                                          00627700
         if ( w < 0.0 ) then k := k + 1 ;                                00627800
         if ( w > 0.0 ) then k1 := k1 + 1 ;                              00627900
         r5 := r5 + w ;                                                  00628000
         w := abs ( w );                                                 00628100
         if ( w > r6 ) then begin                                        00628200
            r6 := w ;                                                    00628300
            x1 := x ;                                                    00628400
         end;                                                            00628500
         r7 := r7 + w * w ;                                              00628600
         x1 := x1 + del ;                                                00628700
      end ;                                                              00628800
                                                                         00628900
      r5 := r5 / xn ;                                                    00629000
      r7 := sqrt ( r7 / xn );                                            00629100
      writeln(' TEST OF EXP(X-', v:7:4, ') VS EXP(X)/EXP(', v:7:4, ')');00629200
      writeln;                                                           00629300
      printtestrun(n,a,b,k,k1,r5,r6,x1,r7);                              00629400
      if ( j = 2 ) then begin                                            00629500
         a := - 2.0 * a ;                                                00629600
         b := 10.0 * a ;                                                 00629700
         if ( b < d ) then b := d ;                                      00629800
      end else begin                                                     00629900
         v := 45.0 / 16.0 ;                                              00630000
```

```
      a := - 10.0 * b ;                                                    00630100
      b := 4.0 * xmin * exp ( it * ln( beta ));                            00630200
      b := ln ( b );                                                       00630300
    end;                                                                   00630400
  end ;                                                                    00630500
{                                                                          00630600
    special tests                                                          00630700
                                                                         } 00630800
  writeln(' THE IDENTITY EXP(X) * EXP(-X) - 1.0 WILL BE TESTED.');         00630900
  writeln(' ':7,'X', ' ':9, 'F(X)*F(-X) - 1');                             00631000
  writeln;                                                                 00631100
                                                                           00631200
  for i := 1 to 5 do begin                                                 00631300
    x := random * beta ;                                                   00631400
    y := - x ;                                                             00631500
    z := exp ( x )* exp ( y )- 1.0 ;                                       00631600
    writeln(x:15, z:15);                                                   00631700
  end ;                                                                    00631800
  writeln;                                                                 00631900
  writeln(' TEST OF SPECIAL ARGUMENTS');                                   00632000
  writeln;                                                                 00632100
  x := 0.0 ;                                                               00632200
  y := exp ( x )- 1.0 ;                                                    00632300
  writeln(' EXP(0.0) - 1.0 = ', y:15);                                     00632400
  writeln;                                                                 00632500
  x := trunc ( ln ( xmin ));                                               00632600
  y := exp ( x );                                                          00632700
  writeln(' EXP(', x:13, ') = ', z:15);                                    00632800
  writeln;                                                                 00632900
  x := trunc ( ln ( xmax ));                                               00633000
  y := exp ( x );                                                          00633100
  writeln(' EXP(', x:13, ') = ', y:15);                                    00633200
  writeln;                                                                 00633300
  x := x / 2.0 ;                                                           00633400
  v := x / 2.0 ;                                                           00633500
  y := exp ( x );                                                          00633600
  z := exp ( v );                                                          00633700
  z := z * z ;                                                             00633800
  writeln(' IF EXP(', x:13, ') = ', y:15, ' IS NOT ABOUT');                00633900
  write(' EXP(', v:13, ')**2 = ', z:15, ' THERE IS AN ARGUMENT');          00634000
  writeln(' REDUCTION ERROR');                                             00634100
  writeln;                                                                 00634200
{                                                                          00634300
    test of error returns                                                  00634400
                                                                         } 00634500
  writeln(' TEST OF ERROR RETURNS');                                       00634600
  writeln;                                                                 00634700
  x := - 1.0 / sqrt ( xmin );                                              00634800
  writeln(' EXP WILL BE CALLED WITH THE ARGUMENT', x:15);                  00634900
  writeln(' THIS SHOULD UNDERFLOW AND MAYBE PRODUCE ZERO OR AN ERROR')     00635000
  ;                                                                        00635100
  writeln;                                                                 00635200
  y := exp ( x );                                                          00635300
  writeln(' EXP RETURNED THE VALUE', y:15);                                00635400
  writeln;                                                                 00635500
  writeln(' THIS CONCLUDES THE TESTS');                                    00635600
end.                                                                       00635700
```

```
{TEST 6.6.6.2-9, CLASS=QUALITY}                                            00635800
                                                                           00635900
{ This test checks the implementation of the sin and cos functions. }      00636000
                                                                           00636100
program t6p6p6p2d9(output);                                                00636200
                                                                           00636300
var                                                                        00636400
                                                                           00636500
                                                                           00636600
{     data required                                                        00636700
                                                                           00636800
        none                                                               00636900
                                                                           00637000
      other subprograms in this package                                    00637100
                                                                           00637200
        machar -  as for sqrtest                                           00637300
        random -  as for sqrtest                                           00637400
                                                                           00637500
                                                                           00637600
      standard subprograms required                                        00637700
                                                                           00637800
        abs, ln, exp, cos, sin, sqrt                                       00637900
                                                                         } 00638000
                                                                           00638100
  i , ibeta , iexp ,  irnd , it , il , j , k , k1 , machep ,               00638200
  iy , maxexp , minexp , n , negep , ngrd : integer ;                      00638300
  a , albeta , b , beta , betap , c , del , eps , epsneg , expon , r5,     00638400
  r6 , r7 , w , x , xl , xmax , xmin , xn , x1 , y , z , zz : real ;       00638500
                                                                           00638600
procedure machar (var ibeta , it , irnd , ngrd , machep , negep , iexp,   00638700
  minexp , maxexp : integer ; var eps , epsneg , xmin , xmax : real ) ;    00638800
                                                                           00638900
var                                                                        00639000
                                                                           00639100
                                                                           00639200
{     This subroutine is intended to determine the characteristics         00639300
      of the floating-point arithmetic system that are specified           00639400
      below.  The first three are determined according to an               00639500
      algorithm due to M. Malcolm, CACM 15 (1972), pp. 949-951,            00639600
      incorporating some, but not all, of the improvements                 00639700
      suggested by M. Gentleman and S. Marovich, CACM 17 (1974),           00639800
      pp. 276-277.  The version given here is for single precision.        00639900
                                                                           00640000
      Latest revision - October 1, 1976.                                   00640100
                                                                           00640200
      Author - W. J. Cody                                                  00640300
               Argonne National Laboratory                                 00640400
                                                                           00640500
      Revised for Pascal - R. A. Freak                                     00640600
                           University of Tasmania                          00640700
                           Hobart                                          00640800
                           Tasmania                                        00640900
                                                                           00641000
      ibeta    is the radix of the floating-point representation           00641100
      it       is the number of base ibeta digits in the floating-point    00641200
               significand                                                 00641300
      irnd     = 0 if the arithmetic chops,                                00641400
               1 if the arithmetic rounds                                  00641500
      ngrd     = 0 if  irnd=1, or if  irnd=0  and only  it  base ibeta     00641600
               digits participate in the post normalization shift          00641700
               of the floating-point significand in multiplication         00641800
```

```
                1 if  irnd=0  and more than  it  base  ibeta  digits  00641900
                  participate in the post normalization shift of the  00642000
                  floating-point significand in multiplication        00642100
      machep    is the exponent on the smallest positive floating-point  00642200
                  number  eps such that  1.0+eps <> 1.0               00642300
      negeps    is the exponent on the smallest positive fl. pt. no.  00642400
                  negeps such that  1.0-negeps <> 1.0, except that    00642500
                  negeps is bounded below by  it-3                    00642600
      iexp      is the number of bits (decimal places if ibeta = 10)  00642700
                  reserved for the representation of the exponent of  00642800
                  a floating-point number                             00642900
      minexp    is the exponent of the smallest positive fl. pt. no.  00643000
                  xmin                                                00643100
      maxexp    is the exponent of the largest finite floating-point  00643200
                  number  xmax                                        00643300
      eps       is the smallest positive floating-point number such   00643400
                  that  1.0+eps <> 1.0. in particular,                00643500
                  eps = ibeta**machep                                 00643600
      epsneg    is the smallest positive floating-point number such   00643700
                  that  1.0-eps <> 1.0  (except that the exponent     00643800
                  negeps is bounded below by it-3).  in particular    00643900
                  epsneg = ibeta**negep                               00644000
      xmin      is the smallest positive floating-point number.  in   00644100
                  particular,  xmin = ibeta ** minexp                 00644200
      xmax      is the largest finite floating-point number.  in      00644300
                  particular   xmax = (1.0-epsneg) * ibeta ** maxexp  00644400
                  note - on some machines  xmax  will be only the     00644500
                  second, or perhaps third, largest number, being     00644600
                  too small by 1 or 2 units in the last digit of      00644700
                  the significand.                                    00644800
                                                                      00644900
                                                                    } 00645000
                                                                      00645100
   i , iz , j , k , mx : integer ;                                    00645200
   a , b , beta , betain , betaml , one , y , z , zero : real ;       00645300
   underflo : boolean;                                                00645400
                                                                      00645500
begin                                                                 00645600
   irnd := 1 ;                                                        00645700
   one := ( irnd );                                                   00645800
   a := one + one ;                                                   00645900
   b := a ;                                                           00646000
   zero := 0.0 ;                                                      00646100
{                                                                     00646200
      determine ibeta,beta ala Malcolm                                00646300
                                                                    } 00646400
   while ( ( ( a + one ) - a ) - one = zero ) do begin                00646500
      a := a + a ;                                                    00646600
   end ;                                                              00646700
   while ( ( a + b ) - a = zero ) do begin                            00646800
      b := b + b ;                                                    00646900
   end ;                                                              00647000
   ibeta := trunc ( ( a + b ) - a );                                  00647100
   beta := ( ibeta );                                                 00647200
   betaml := beta - one ;                                             00647300
{                                                                     00647400
      determine irnd,ngrd,it                                          00647500
                                                                    } 00647600
   if ( ( a + betaml ) - a = zero ) then irnd := 0 ;                  00647700
   it := 0 ;                                                          00647800
   a := one ;                                                         00647900
   repeat begin                                                       00648000
      it := it + 1 ;                                                  00648100
      a := a * beta ;                                                 00648200
   end until ( ( ( a + one ) - a ) - one <> zero ) ;                  00648300
{                                                                     00648400
      determine negep, epsneg                                         00648500
                                                                    } 00648600
   negep := it + 3 ;                                                  00648700
   a := one ;                                                         00648800
                                                                      00648900
   for i := 1 to negep do begin                                       00649000
      a := a / beta ;                                                 00649100
   end ;                                                              00649200
                                                                      00649300
   while ( ( one - a ) - one = zero ) do begin                        00649400
      a := a * beta ;                                                 00649500
      negep := negep - 1 ;                                            00649600
   end ;                                                              00649700
   negep := - negep ;                                                 00649800
   epsneg := a ;                                                      00649900
{                                                                     00650000
      determine machep, eps                                           00650100
                                                                    } 00650200
   machep := negep ;                                                  00650300
   while ( ( one + a ) - one = zero ) do begin                        00650400
      a := a * beta ;                                                 00650500
      machep := machep + 1 ;                                          00650600
   end ;                                                              00650700
   eps := a ;                                                         00650800
{                                                                     00650900
      determine ngrd                                                  00651000
                                                                    } 00651100
   ngrd := 0 ;                                                        00651200
   if(( irnd = 0) and((( one + eps) * one - one) <> zero)) then       00651300
   ngrd := 1 ;                                                        00651400
{                                                                     00651500
      determine iexp, minexp, xmin                                    00651600
                                                                      00651700
      loop to determine largest i such that                           00651800
          (1/beta) ** (2**(i))                                        00651900
      does not underflow                                              00652000
      exit from loop is signall by an underflow                       00652100
                                                                    } 00652200
   i := 0 ;                                                           00652300
   betain := one / beta ;                                             00652400
   z := betain ;                                                      00652500
   underflo := false;                                                 00652600
   repeat begin                                                       00652700
      y := z ;                                                        00652800
      z := y * y ;                                                    00652900
{                                                                     00653000
      check for underflow                                             00653100
                                                                    } 00653200
      if ( ( z * one = zero ) or ( abs ( z )> y ) ) then begin        00653300
         underflo := true;                                            00653400
      end else begin                                                  00653500
         i := i + 1 ;                                                 00653600
      end;                                                            00653700
   end until underflo ;                                               00653800
   k := 1 ;                                                           00653900
{                                                                     00654000
```

```
   determine k such that (1/beta)**k does not underflow

   first set k = 2 ** i


for j := 1 to i do begin
   k := k + k ;
end ;

iexp := i + 1 ;
mx := k + k ;
if ( ibeta = 10 ) then begin
{
   for decimal machines only
   iexp := 2 ;
   iz := ibeta ;
   while ( k >= iz ) do begin
      iz := iz * ibeta ;
      iexp := iexp + 1 ;
   end ;
   mx := iz + iz - 1 ;
end;
underflo := false;
repeat begin
{
   loop to construct xmin
   exit from loop is signalled by an underflow

   xmin := y ;
   y := y * betain ;
   if ( ( ( y * one ) = zero ) or ( abs ( y )> xmin ) )
   then begin
      underflo := true;
   end else begin
      k := k + 1 ;
   end;
end until underflo ;
minexp := - k ;
{  determine maxexp, xmax

if ( ( mx <= k + k - 3 ) and ( ibeta <> 10 ) ) then begin
   mx := mx + mx ;
   iexp := iexp + 1 ;
end;
maxexp := mx + minexp ;
{  adjust for machines with implicit leading
   bit in binary significand and machines with
   radix point at extreme right of significand

i := maxexp + minexp ;
if ( ( ibeta = 2 ) and ( i = 0 ) ) then maxexp := maxexp - 1 ;
if ( i > 20 ) then maxexp := maxexp - 3 ;
xmax := one - epsneg ;
if ( xmax * one <> xmax ) then xmax := one - beta * epsneg ;
xmax := ( xmax * betain * betain * betain ) / xmin ;
i := maxexp + minexp + 3 ;
if  ( i > 0 ) then begin

   for j := 1 to i do begin
      xmax := xmax * beta ;
   end ;
```

```
   end;

end;


function random : real ;

{     random number generator - based on algorithm 266
       by Pike and Hill (modified by Hansson)
       collected Alg. from CACM.

      This subprogram is intended for use on computers with
       fixed point wordlength of at least 29 bIts.  it is
       best if the floating point significand has at most
       29 bits. }

{     The quality of the random numbers is not important.
      If recoding is needed for small wordlength computers,
      even returning a constant value or zero is possible. }

{     The value iy is global, and is initialized in the driver }

begin
   iy := (iy*125) mod 2796203;
   random := ( iy )/ 2796203.0e0 ;
end;


procedure printtestrun (n:integer; lb,ub:real;
                        big,small : integer;
                        mean,maxerror,xmaxerror,rmserror:real);
begin
   writeln(' ':5,n:4,' RANDOM ARGUMENTS WERE TESTED FROM THE INTERVAL')
      ;
   writeln(' ':10,'(',lb:15,',',ub:15,')');
   writeln;
   writeln(' ':5,'THE RESULT WAS TOO LARGE',big:5,' TIMES, AND');
   writeln(' ':10,'TOO SMALL',small:5,' TIMES');
   writeln;
   if (mean <> 0.0) then begin
      writeln(' ':5,'MEAN RELATIVE ERROR =',mean:15,'=',
         IBETA:4,' ** ',LN(ABS(mean))/ALBETA:7:2);
   end;
   if (maxerror<> 0.0) then begin
      writeln(' ':5,'THE MAXIMUM RELATIVE ERROR OF',maxerror:15,'=',
         IBETA:4,' ** ',LN(ABS(maxerror))/ALBETA:7:2);
      writeln(' ':10,'OCCURRED FOR X =',xmaxerror:15);
   end;
   if (rmserror <> 0.0) then begin
      writeln(' ':5,'ROOT-MEAN-SQUARE RELATIVE ERROR =',rmserror:15,
         '=',IBETA:4,' ** ',LN(ABS(rmserror))/ALBETA:7:2);
   end;
   writeln;
end;   { OF PRINT TEST RUN }


begin
   iy := 100001;
   machar ( ibeta , it , irnd , ngrd , machep , negep , iexp , minexp ,
      maxexp , eps , epsneg , xmin , xmax );
   beta := ( ibeta );
```

```
   albeta := ln ( beta );                                                    00666300
   a := 0.0 ;                                                                00666400
   b := 1.570796327 ;                                                        00666500
   c := b ;                                                                  00666600
   n := 2000 ;                                                               00666700
   xn := ( n );                                                              00666800
   il := 0 ;                                                                 00666900
{                                                                            00667000
      random argument accuracy tests                                         00667100
}                                                                            00667200
   for j := 1 to 3 do begin                                                  00667300
      k := 0 ;                                                               00667400
      k1 := 0 ;                                                              00667500
      x1 := 0.0 ;                                                            00667600
      r5 := 0.0 ;                                                            00667700
      r6 := 0.0 ;                                                            00667800
      r7 := 0.0 ;                                                            00667900
      del := ( b - a ) / xn ;                                                00668000
      x1 := a ;                                                              00668100
                                                                             00668200
      for i := 1 to n do begin                                               00668300
         x := del * random + x1 ;                                            00668400
         y := x / 3.0 ;                                                      00668500
         y := ( x + y ) - x ;                                                00668600
         x := 3.0 * y ;                                                      00668700
         if ( j = 3 ) then begin                                             00668800
            z := cos ( x );                                                  00668900
            zz := cos ( y );                                                 00669000
            w := ( z + zz * ( 3.0 - 4.0 * zz * zz ) ) / z ;                  00669100
         end else begin                                                      00669200
            z := sin ( x );                                                  00669300
            zz := sin ( y );                                                 00669400
            w := ( z - zz * ( 3.0 - 4.0 * zz * zz ) ) / z ;                  00669500
         end;                                                                00669600
         if ( w > 0.0 ) then k := k + 1 ;                                    00669700
         if ( w < 0.0 ) then k1 := k1 + 1 ;                                  00669800
         r5 := r5 + w ;                                                      00669900
         w := abs ( w );                                                     00670000
         if ( w > r6 ) then begin                                            00670100
            r6 := w ;                                                        00670200
            x1 := x ;                                                        00670300
         end;                                                                00670400
         r7 := r7 + w * w ;                                                  00670500
         x1 := x1 + del ;                                                    00670600
      end ;                                                                  00670700
                                                                             00670800
      r5 := r5 / xn ;                                                        00670900
      r7 := sqrt ( r7 / xn );                                                00671000
      if ( j = 3 ) then begin                                                00671100
         writeln(' TEST OF COS(X) VS 4*COS(X/3)**3-3*COS(X/3)');             00671200
         writeln;                                                            00671300
      end else begin                                                         00671400
         writeln(' TEST OF SIN(X) VS 3*SIN(X/3)-4*SIN(X/3)**3');             00671500
         writeln;                                                            00671600
      end;                                                                   00671700
      printtestrun(n,a,b,k,k1,r5,r6,x1,r7);                                  00671800
      a := 18.84955592 ;                                                     00671900
      if ( j = 2 ) then a := b + c ;                                         00672000
      b := a + c ;                                                           00672100
   end ;                                                                     00672200
{                                                                            00672300
```

```
      special tests                                                          00672400
}                                                                            00672500
   c := 1.0 / exp ( ( it div 2 ) * ln( beta ));                              00672600
   z := ( sin ( a + c )- sin ( a - c )) / ( c + c ) ;                        00672700
   write(' IF ', z:15,' IS NOT ALMOST 1.0 THEN SIN HAS THE WRONG ');         00672800
   writeln('PERIOD');                                                        00672900
   writeln;                                                                  00673000
                                                                             00673100
   writeln(' THE IDENTITY -SIN(X) = -SIN(X) WILL BE TESTED');                00673200
   writeln(' ':7, 'X',' ':9,'F(X) + F(-X)');                                 00673300
   writeln;                                                                  00673400
                                                                             00673500
   for i := 1 to 5 do begin                                                  00673600
      x := random * a ;                                                      00673700
      z := sin ( x )+ sin ( - x );                                           00673800
      writeln(x:14,z:15);                                                    00673900
   end ;                                                                     00674000
                                                                             00674100
   writeln;                                                                  00674200
   writeln(' THE IDENTITY SIN(X) = X, X SMALL, WILL BE TESTED.');            00674300
   writeln(' ':7, 'X',' ':9,'X - F(X)');                                     00674400
   writeln;                                                                  00674500
   betap := exp ( it * ln( beta ));                                          00674600
   x := random / betap ;                                                     00674700
                                                                             00674800
   for i := 1 to 5 do begin                                                  00674900
      z := x - sin ( x );                                                    00675000
      writeln(x:14, z:15);                                                   00675100
      x := x / beta ;                                                        00675200
   end ;                                                                     00675300
   writeln;                                                                  00675400
                                                                             00675500
   writeln(' THE IDENTITY COS(-X) = COS(X) WILL BE TESTED.');                00675600
   writeln(' ':7,'X',' ':9,'F(X) - F(-X)');                                  00675700
   writeln;                                                                  00675800
                                                                             00675900
   for i := 1 to 5 do begin                                                  00676000
      x := random * a ;                                                      00676100
      z := cos ( x )- cos ( - x );                                           00676200
      writeln(x:14, z:15);                                                   00676300
   end ;                                                                     00676400
   writeln;                                                                  00676500
                                                                             00676600
   writeln(' TEST OF UNDERFLOW FOR VERY SMALL ARGUMENTS');                   00676700
   writeln;                                                                  00676800
   expon := ( minexp )* 0.75 ;                                               00676900
   x := exp ( expon * ln( beta ));                                           00677000
   y := sin ( x );                                                           00677100
   writeln(' ':5, 'SIN(', x:15, ') = ', y:15);                               00677200
   writeln;                                                                  00677300
   writeln(' THE FOLLOWING THREE LINES ILLUSTRATE THE LOSS IN');             00677400
   writeln(' SIGNIFICANCE FOR LARGE ARGUMENTS. THE ARGUMENTS');              00677500
   writeln(' USED ARE CONSECUTIVE.');                                        00677600
   writeln;                                                                  00677700
   z := sqrt ( betap );                                                      00677800
   x := z * ( 1.0 - epsneg ) ;                                               00677900
   y := sin ( x );                                                           00678000
   writeln(' ':5, 'SIN(', x:15, ') = ', y:15);                               00678100
   writeln;                                                                  00678200
   y := sin ( z );                                                           00678300
   writeln(' ':5, 'SIN(', z:15, ') = ', y:15);                               00678400
```

```
   writeln;                                                                  00678500
   x := z * ( 1.0 + eps ) ;                                                  00678600
   y := sin ( x );                                                           00678700
   writeln(' ':5, 'SIN(', x:15, ') = ', y:15);                               00678800
   writeln;                                                                  00678900
   x := betap ;                                                              00679000
   writeln(' SIN(X) WILL BE CALLED WITH THE ARGUMENT ', x:15);               00679100
   y := sin ( x );                                                           00679200
   writeln(' SIN RETURNED THE VALUE ', y:15);                                00679300
   writeln(' THIS CONCLUDES THE TESTS.');                                    00679400
end.                                                                         00679500
```

```
{TEST 6.6.6.2-10, CLASS=QUALITY}                                             00679600
                                                                             00679700
{ This test checks the implementation of the ln function. }                  00679800
                                                                             00679900
program t6p6p6p2d10(output);                                                 00680000
                                                                             00680100
var                                                                          00680200
                                                                             00680300
                                                                             00680400
{     data required                                                          00680500
                                                                             00680600
         none                                                                00680700
                                                                             00680800
      other subprograms in this package                                      00680900
                                                                             00681000
         machar -  as for sqrtest                                            00681100
         ran(k) - as for sqrtest                                             00681200
                                                                             00681300
                                                                             00681400
      standard subprograms required                                          00681500
                                                                             00681600
         abs, ln, sqrt                                                       00681700
                                                                           } 00681800
                                                                             00681900
   i , ibeta , iexp ,  irnd , it , il , j , k , kl , machep ,                00682000
     iy , maxexp , minexp , n , negep , ngrd : integer ;                     00682100
   a , albeta , b , beta , d , del , eight , eps , epsneg , half , r5 ,      00682200
     r6 , r7 , tenth , w , x , xl , xmax , xmin , xn , xl , y , z ,zz:       00682300
     real ;                                                                  00682400
                                                                             00682500
procedure machar (var ibeta , it , irnd , ngrd , machep , negep , iexp,     00682600
  minexp , maxexp : integer ; var eps , epsneg , xmin , xmax : real ) ;      00682700
                                                                             00682800
var                                                                          00682900
                                                                             00683000
                                                                             00683100
{     This subroutine is intended to determine the characteristics           00683200
      of the floating-point arithmetic system that are specified             00683300
      below.  The first three are determined according to an                 00683400
      algorithm due to M. Malcolm, CACM 15 (1972), pp. 949-951,              00683500
      incorporating some, but not all, of the improvements                   00683600
      suggested by M. Gentleman and S. Marovich, CACM 17 (1974),             00683700
      pp. 276-277.  The version given here is for single precision.          00683800
                                                                             00683900
      Latest revision - October 1, 1976.                                     00684000
                                                                             00684100
      Author - W. J. Cody                                                    00684200
               Argonne National Laboratory                                   00684300
                                                                             00684400
      Revised for Pascal - R. A. Freak                                       00684500
                           University of Tasmania                            00684600
                           Hobart                                            00684700
                           Tasmania                                          00684800
                                                                             00684900
      ibeta    is the radix of the floating-point representation             00685000
      it       is the number of base ibeta digits in the floating-point      00685100
               significand                                                   00685200
      irnd     =  0 if the arithmetic chops,                                 00685300
               1 if the arithmetic rounds                                    00685400
      ngrd     =  0 if  irnd=1, or if  irnd=0  and only  it  base ibeta      00685500
               digits participate in the post normalization shift            00685600
```

```
                of the floating-point significand in multiplication
              1 if  irnd=0  and more than  it  base  ibeta  digits
                participate in the post normalization shift of the
                floating-point significand in multiplication
     machep   is the exponent on the smallest positive floating-point
              number  eps such that  1.0+eps <> 1.0
     negeps   is the exponent on the smallest positive fl. pt. no.
              negeps such that  1.0-negeps <> 1.0, except that
              negeps is bounded below by  it-3
     iexp     is the number of bits (decimal places if ibeta = 10)
              reserved for the representation of the exponent of
              a floating-point number
     minexp   is the exponent of the smallest positive fl. pt. no.
              xmin
     maxexp   is the exponent of the largest finite floating-point
              number  xmax
     eps      is the smallest positive floating-point number such
              that  1.0+eps <> 1.0. in particular,
              eps = ibeta**machep
     epsneg   is the smallest positive floating-point number such
              that  1.0-eps <> 1.0  (except that the exponent
              negeps is bounded below by it-3).  in particular
              epsneg = ibeta**negep
     xmin     is the smallest positive floating-point number.  in
              particular,  xmin = ibeta ** minexp
     xmax     is the largest finite floating-point number.  in
              particular   xmax = (1.0-epsneg) * ibeta ** maxexp
              note - on some machines  xmax  will be only the
              second, or perhaps third, largest number, being
              too small by 1 or 2 units in the last digit of
              the significand.
                                                                  }

   i , iz , j , k , mx : integer ;
   a , b , beta , betain , betaml , one , y , z , zero : real ;
   underflo : boolean;

begin
   irnd := 1 ;
   one := ( irnd );
   a := one + one ;
   b := a ;
   zero := 0.0 ;
{
      determine ibeta,beta ala Malcolm
                                                                  }
   while ( ( ( a + one ) - a ) - one = zero ) do begin
      a := a + a ;
   end ;
   while ( ( a + b ) - a = zero ) do begin
      b := b + b ;
   end ;
   ibeta := trunc ( ( a + b ) - a );
   beta := ( ibeta );
   betaml := beta - one ;
{
      determine irnd,ngrd,it
                                                                  }
   if ( ( a + betaml ) - a = zero ) then irnd := 0 ;
   it := 0 ;
   a := one ;
   repeat begin
      it := it + 1 ;
      a := a * beta ;
   end until ( ( ( a + one ) - a ) - one <> zero ) ;
{
      determine negep, epsneg
                                                                  }
   negep := it + 3 ;
   a := one ;

   for i := 1 to negep do begin
      a := a / beta ;
   end ;

   while ( ( one - a ) - one = zero ) do begin
      a := a * beta ;
      negep := negep - 1 ;
   end ;
   negep := - negep ;
   epsneg := a ;
{
      determine machep, eps
                                                                  }
   machep := negep ;
   while ( ( one + a ) - one = zero ) do begin
      a := a * beta ;
      machep := machep + 1 ;
   end ;
   eps := a ;
{
      determine ngrd
                                                                  }
   ngrd := 0 ;
   if(( irnd = 0) and((( one + eps) * one - one) <> zero)) then
   ngrd := 1 ;
{
      determine iexp, minexp, xmin

      loop to determine largest i such that
          (1/beta) ** (2**(i))
      does not underflow
      exit from loop is signall by an underflow
                                                                  }
   i := 0 ;
   betain := one / beta ;
   z := betain ;
   underflo := false;
   repeat begin
      y := z ;
      z := y * y ;
{
      check for underflow
                                                                  }
      if ( ( z * one = zero ) or ( abs ( z )> y ) ) then begin
         underflo := true;
      end else begin
         i := i + 1 ;
      end;
   end until underflo ;
   k := 1 ;
```

```
{                                                                    00697900
    determine k such that (1/beta)**k does not underflow             00698000
                                                                     00698100
    first set k = 2 ** i                                             00698200
                                                                  }  00698300
                                                                     00698400
  for j := 1 to i do begin                                           00698500
    k := k + k ;                                                     00698600
  end ;                                                              00698700
                                                                     00698800
  iexp := i + 1 ;                                                    00698900
  mx := k + k ;                                                      00699000
  if ( ibeta = 10 ) then begin                                       00699100
{                                                                    00699200
    for decimal machines only                                     }  00699300
    iexp := 2 ;                                                      00699400
    iz := ibeta ;                                                    00699500
    while ( k >= iz ) do begin                                       00699600
      iz := iz * ibeta ;                                             00699700
      iexp := iexp + 1 ;                                             00699800
    end ;                                                            00699900
    mx := iz + iz - 1 ;                                              00700000
  end;                                                               00700100
  underflo := false;                                                 00700200
  repeat begin                                                       00700300
{                                                                    00700400
    loop to construct xmin                                           00700500
    exit from loop is signalled by an underflow                      00700600
                                                                  }  00700700
    xmin := y ;                                                      00700800
    y := y * betain ;                                                00700900
    if ( ( ( y * one ) = zero ) or ( abs ( y )> xmin ) )             00701000
    then begin                                                       00701100
      underflo := true;                                              00701200
    end else begin                                                   00701300
      k := k + 1 ;                                                   00701400
    end;                                                             00701500
  end until underflo ;                                               00701600
  minexp := - k ;                                                    00701700
{ determine maxexp, xmax                                             00701800
                                                                  }  00701900
  if ( ( mx <= k + k - 3 ) and ( ibeta <> 10 ) ) then begin          00702000
    mx := mx + mx ;                                                  00702100
    iexp := iexp + 1 ;                                               00702200
  end;                                                               00702300
  maxexp := mx + minexp ;                                            00702400
{ adjust for machines with implicit leading                          00702500
  bit in binary significand and machines with                        00702600
  radix point at extreme right of significand                        00702700
                                                                  }  00702800
  i := maxexp + minexp ;                                             00702900
  if ( ( ibeta = 2 ) and ( i = 0 ) ) then maxexp := maxexp - 1 ;     00703000
  if ( i > 20 ) then maxexp := maxexp - 3 ;                          00703100
  xmax := one - epsneg ;                                             00703200
  if ( xmax * one <> xmax ) then xmax := one - beta * epsneg ;       00703300
  xmax := ( xmax * betain * betain * betain ) / xmin ;               00703400
  i := maxexp + minexp + 3 ;                                         00703500
  if  ( i > 0 ) then begin                                           00703600
                                                                     00703700
    for j := 1 to i do begin                                         00703800
      xmax := xmax * beta ;                                          00703900
    end ;                                                            00704000
  end;                                                               00704100
                                                                     00704200
end;                                                                 00704300
                                                                     00704400
                                                                     00704500
function random : real ;                                             00704600
                                                                     00704700
{     random number generator - based on algorithm 266               00704800
       by Pike and Hill (modified by Hansson)                        00704900
       collected Alg. from CACM.                                     00705000
                                                                     00705100
      This subprogram is intended for use on computers with          00705200
       fixed point wordlength of at least 29 bIts.  it is            00705300
       best if the floating point significand has at most            00705400
       29 bits. }                                                    00705500
                                                                     00705600
{     The quality of the random numbers is not important.            00705700
      If recoding is needed for small wordlength computers,          00705800
      even returning a constant value or zero is possible. }         00705900
                                                                     00706000
{     The value iy is global, and is initialized in the driver }     00706100
                                                                     00706200
begin                                                                00706300
  iy := (iy*125) mod 2796203;                                        00706400
  random := ( iy )/ 2796203.0e0 ;                                    00706500
end;                                                                 00706600
                                                                     00706700
                                                                     00706800
procedure printtestrun (n:integer; lb,ub:real;                       00706900
                        big,small : integer;                         00707000
                        mean,maxerror,xmaxerror,rmserror:real);      00707100
begin                                                                00707200
  writeln(' ':5,n:4,' RANDOM ARGUMENTS WERE TESTED FROM THE INTERVAL')00707300
    ;                                                                00707400
  writeln(' ':10,'(',lb:15,',',ub:15,')');                           00707500
  writeln;                                                           00707600
  writeln(' ':5,'THE RESULT WAS TOO LARGE',big:5,' TIMES, AND');     00707700
  writeln(' ':10,'TOO SMALL',small:5,' TIMES');                      00707800
  writeln;                                                           00707900
  if (mean <> 0.0) then begin                                        00708000
    writeln(' ':5,'MEAN RELATIVE ERROR =',mean:15,'=',               00708100
      IBETA:4,' ** ',LN(ABS(mean))/ALBETA:7:2);                      00708200
  end;                                                               00708300
  if (maxerror<> 0.0) then begin                                     00708400
    writeln(' ':5,'THE MAXIMUM RELATIVE ERROR OF',maxerror:15,'=',   00708500
      IBETA:4,' ** ',LN(ABS(maxerror))/ALBETA:7:2);                  00708600
    writeln(' ':10,'OCCURRED FOR X =',xmaxerror:15);                 00708700
  end;                                                               00708800
  if (rmserror <> 0.0) then begin                                    00708900
    writeln(' ':5,'ROOT-MEAN-SQUARE RELATIVE ERROR =',rmserror:15,   00709000
      '=',IBETA:4,' ** ',LN(ABS(rmserror))/ALBETA:7:2);              00709100
  end;                                                               00709200
  writeln;                                                           00709300
end;   { OF PRINT TEST RUN }                                        00709400
                                                                     00709500
function sign(a1 , a2 : real) : real;                                00709600
begin                                                                00709700
  if (a2 < 0) then                                                   00709800
    sign := -abs(a1)                                                 00709900
  else                                                               00710000
```

```
    sign := abs(a1)                                                          00710100
end;                                                                         00710200
                                                                             00710300
                                                                             00710400
begin                                                                        00710500
   iy := 100001;                                                             00710600
   machar ( ibeta , it , irnd , ngrd , machep , negep , iexp , minexp ,      00710700
     maxexp , eps , epsneg , xmin , xmax );                                  00710800
   beta := ( ibeta );                                                        00710900
   albeta := ln ( beta );                                                    00711000
   j := it div 3 ;                                                           00711100
   a := 1.0 ;                                                                00711200
                                                                             00711300
   for i := 1 to j do begin                                                  00711400
      a := a / beta ;                                                        00711500
   end ;                                                                     00711600
                                                                             00711700
   n := 2000 ;                                                               00711800
   xn := ( n );                                                              00711900
   b := 1.0 + a ;                                                            00712000
   a := 1.0 - a ;                                                            00712100
   d := 1.0 + sqrt ( eps );                                                  00712200
   half := 0.5 ;                                                             00712300
   eight := 8.0 ;                                                            00712400
   tenth := 0.1 ;                                                            00712500
   i1 := 0 ;                                                                 00712600
{                                                                            00712700
      random argument accuracy tests                                         00712800
                                                                           } 00712900
   for j := 1 to 4 do begin                                                  00713000
      case j of                                                              00713100
      1:                                                                     00713200
         begin                                                               00713300
            write(' TEST OF LN(X) VS TAYLOR SERIES EXPANSION');               00713400
            writeln(' OF LN(1+Y)');                                          00713500
         end;                                                                00713600
      2:                                                                     00713700
         begin                                                               00713800
            writeln(' TEST OF LN(X) VS LN(17X/16)-LN(17/16)');                00713900
            writeln;                                                         00714000
            a := sqrt ( half );                                              00714100
            b := 15.0 / 16.0 ;                                               00714200
            end;                                                             00714300
      3:                                                                     00714400
         begin                                                               00714500
            writeln(' TEST OF LN(X) VS LN(11X/10)-LN(11/10)');                00714600
            writeln;                                                         00714700
            a := sqrt ( tenth );                                             00714800
            b := 0.9;                                                        00714900
         end;                                                                00715000
      4:                                                                     00715100
         begin                                                               00715200
            writeln(' TEST OF LN(X*X) VS 2 * LN(X)');                        00715300
            writeln;                                                         00715400
            a := 16.0 ;                                                      00715500
            b := 240.0 ;                                                     00715600
         end;                                                                00715700
      end;                                                                   00715800
      del := ( b - a ) / xn ;                                                00715900
      k := 0 ;                                                               00716000
      k1 := 0 ;                                                              00716100
      x1 := 0.0 ;                                                            00716200
      r5 := 0.0 ;                                                            00716300
      r6 := 0.0 ;                                                            00716400
      r7 := 0.0 ;                                                            00716500
      x1 := a ;                                                              00716600
                                                                             00716700
      for i := 1 to n do begin                                               00716800
         x := del * random * d + x1 ;                                        00716900
         case j of                                                           00717000
         1:                                                                  00717100
            begin                                                            00717200
               y := ( x - 0.5 ) - half ;                                     00717300
               z := ln ( x );                                                00717400
               zz := 1.0 / 3.0 ;                                             00717500
               zz := y * ( zz - y / 4.0 ) ;                                  00717600
               zz := ( zz - half ) * y * y + y ;                             00717700
            end;                                                             00717800
         2:                                                                  00717900
            begin                                                            00718000
               x := ( x + 8.0 ) - eight ;                                    00718100
               y := x + x / 16.0 ;                                           00718200
               z := ln ( x );                                                00718300
               zz := ln ( y )- 7.7746816434842581e-5 ;                       00718400
               zz := zz - 31.0 / 512.0 ;                                     00718500
            end;                                                             00718600
         3:                                                                  00718700
            begin                                                            00718800
               x := ( x + 8.0 ) - eight ;                                    00718900
               y := x + x * tenth ;                                          00719000
               z := ln ( x )/ ln(10) ;                                       00719100
               zz := ln ( y )/ ln(10) - 3.7706015822504075e-4 ;              00719200
               zz := zz - 21.0 / 512.0 ;                                     00719300
            end;                                                             00719400
         4:                                                                  00719500
            begin                                                            00719600
               z := ln ( x * x );                                            00719700
               zz := ln ( x );                                               00719800
               zz := zz + zz ;                                               00719900
            end;                                                             00720000
         end;                                                                00720100
         w := ( z - zz ) / z ;                                               00720200
         z := sign ( w , z );                                                00720300
         if ( z > 0.0 ) then k := k + 1 ;                                    00720400
         if ( z < 0.0 ) then k1 := k1 + 1 ;                                  00720500
         r5 := r5 + w ;                                                      00720600
         w := abs ( w );                                                     00720700
         if ( w > r6 ) then begin                                            00720800
            r6 := w ;                                                        00720900
            x1 := x ;                                                        00721000
         end;                                                                00721100
         r7 := r7 + w * w ;                                                  00721200
         x1 := x1 + del ;                                                    00721300
      end ;                                                                  00721400
                                                                             00721500
      r5 := r5 / xn ;                                                        00721600
      r7 := sqrt ( r7 / xn );                                                00721700
      printtestrun(n,a,b,k,k1,r5,r6,x1,r7);                                  00721800
   end ;                                                                     00721900
{                                                                            00722000
   special tests                                                             00722100
                                                                           } 00722200
```

```
   writeln(' THE IDENTITY LN(X) = - LN(1/X) WILL BE TESTED');
   writeln;
   writeln('        X        F(X) + F(1/X)');
   writeln;

   for i := 1 to 5 do begin
      x := random ;
      x := x + x + 15.0 ;
      y := 1.0 / x ;
      z := ln ( x )+ ln ( y );
      writeln(x:15,z:15);
   end ;

   writeln;
   writeln(' TEST OF SPECIAL ARGUMENTS');
   writeln;
   x := 1.0 ;
   y := ln ( x );
   writeln(' LN(1.0) = ', y:15);
   writeln;
   x := xmin ;
   y := ln ( x );
   writeln(' LN(XMIN) = LN(', x:15, ') = ', y:15);
   writeln;
   x := xmax ;
   y := ln ( x );
   writeln(' LN(XMAX) = LN(', x:15, ') = ', y:15);
   writeln;
{
      Test 6.6.6.2-4 checks that an error is produced
      when  ln is called with a negative argument.
                                                        }
   writeln(' THIS CONCLUDES THE TESTS');
end.
```

```
{TEST 6.6.6.2-11, CLASS=IMPLEMENTATIONDEFINED}

{ This program determines some of the characteristics of the
  floating-point arithmetic system of the host machine.
  If the program fails or the printed results do not agree
  with the known data for the machine then the program
  should be checked because some of the assumptions made
  about floating-point arithmetic may be invalid for that
  machine. }

program t6p6p6p2d11(output);

{ If the results from this test are not in conformity with
  the known data for the implementation,  then all copies of
  MACHAR should be replaced by an equivalent that assigns
  the appropriate values to the parameters. }

var

   eps , epsneg , xmax , xmin : real;

   ibeta , iexp , irnd , it , machep , maxexp , minexp , negep , ngrd :
   integer;

procedure machar (var ibeta , it , irnd , ngrd , machep , negep , iexp,
  minexp , maxexp : integer ; var eps , epsneg , xmin , xmax : real ) ;

var


{     This subroutine is intended to determine the characteristics
      of the floating-point arithmetic system that are specified
      below.  The first three are determined according to an
      algorithm due to M. Malcolm, CACM 15 (1972), pp. 949-951,
      incorporating some, but not all, of the improvements
      suggested by M. Gentleman and S. Marovich, CACM 17 (1974),
      pp. 276-277.  The version given here is for single precision.

      Latest revision - October 1, 1976.

      Author - W. J. Cody
               Argonne National Laboratory

      Revised for Pascal - R. A. Freak
                           University of Tasmania
                           Hobart
                           Tasmania

      ibeta    is the radix of the floating-point representation
      it       is the number of base ibeta digits in the floating-point
               significand
      irnd     =  0 if the arithmetic chops,
                  1 if the arithmetic rounds
      ngrd     =  0 if  irnd=1, or if  irnd=0  and only  it  base ibeta
                  digits participate in the post normalization shift
                  of the floating-point significand in multiplication
                  1 if  irnd=0  and more than  it  base  ibeta  digits
                  participate in the post normalization shift of the
                  floating-point significand in multiplication
      machep   is the exponent on the smallest positive floating-point
               number  eps such that  1.0+eps <> 1.0
```

```
    negeps   is the exponent on the smallest positive fl. pt. no.
                negeps such that  1.0-negeps <> 1.0, except that
                negeps is bounded below by  it-3
    iexp     is the number of bits (decimal places if ibeta = 10)
                reserved for the representation of the exponent of
                a floating-point number
    minexp   is the exponent of the smallest positive fl. pt. no.
                xmin
    maxexp   is the exponent of the largest finite floating-point
                number  xmax
    eps      is the smallest positive floating-point number such
                that  1.0+eps <> 1.0. in particular,
                eps = ibeta**machep
    epsneg   is the smallest positive floating-point number such
                that  1.0-eps <> 1.0  (except that the exponent
                negeps is bounded below by it-3).  in particular
                epsneg = ibeta**negep
    xmin     is the smallest positive floating-point number.  in
                particular,  xmin = ibeta ** minexp
    xmax     is the largest finite floating-point number.  in
                particular   xmax = (1.0-epsneg) * ibeta ** maxexp
                note - on some machines  xmax  will be only the
                second, or perhaps third, largest number, being
                too small by 1 or 2 units in the last digit of
                the significand.
                                                                  }

  i , iz , j , k , mx : integer ;
  a , b , beta , betain , betaml , one , y , z , zero : real ;
  underflo : boolean;

begin
  irnd := 1 ;
  one := ( irnd );
  a := one + one ;
  b := a ;
  zero := 0.0 ;
{
    determine ibeta,beta ala Malcolm
                                                                  }
  while ( ( ( a + one ) - a ) - one = zero ) do begin
    a := a + a ;
  end ;
  while ( ( a + b ) - a = zero ) do begin
    b := b + b ;
  end ;
  ibeta := trunc ( ( a + b ) - a );
  beta := ( ibeta );
  betaml := beta - one ;
{
    determine irnd,ngrd,it
                                                                  }
  if ( ( a + betaml ) - a = zero ) then irnd := 0 ;
  it := 0 ;
  a := one ;
  repeat begin
    it := it + 1 ;
    a := a * beta ;
  end until ( ( ( a + one ) - a ) - one <> zero ) ;
{
    determine negep, epsneg                                       }
  negep := it + 3 ;
  a := one ;

  for i := 1 to negep do begin
    a := a / beta ;
  end ;

  while ( ( one - a ) - one = zero ) do begin
    a := a * beta ;
    negep := negep - 1 ;
  end ;
  negep := - negep ;
  epsneg := a ;
{
    determine machep, eps
                                                                  }
  machep := negep ;
  while ( ( one + a ) - one = zero ) do begin
    a := a * beta ;
    machep := machep + 1 ;
  end ;
  eps := a ;
{
    determine ngrd
                                                                  }
  ngrd := 0 ;
  if(( irnd = 0) and((( one + eps) * one - one) <> zero)) then
  ngrd := 1 ;
{
    determine iexp, minexp, xmin

    loop to determine largest i such that
        (1/beta) ** (2**(i))
    does not underflow
    exit from loop is signall by an underflow

  i := 0 ;
  betain := one / beta ;
  z := betain ;
  underflo := false;
  repeat begin
    y := z ;
    z := y * y ;
{
    check for underflow

    if ( ( z * one = zero ) or ( abs ( z )> y ) ) then begin
      underflo := true;
    end else begin
      i := i + 1 ;
    end;
  end until underflo ;
  k := 1 ;
{
    determine k such that (1/beta)**k does not underflow

    first set k = 2 ** i
                                                                  }
```

```
   for j := 1 to i do begin                                                  00744000
      k := k + k ;                                                           00744100
   end ;                                                                     00744200
                                                                             00744300
   iexp := i + 1 ;                                                           00744400
   mx := k + k ;                                                             00744500
   if ( ibeta = 10 ) then begin                                              00744600
{                                                                            00744700
      for decimal machines only                                           } 00744800
      iexp := 2 ;                                                            00744900
      iz := ibeta ;                                                          00745000
      while ( k >= iz ) do begin                                             00745100
         iz := iz * ibeta ;                                                  00745200
         iexp := iexp + 1 ;                                                  00745300
      end ;                                                                  00745400
      mx := iz + iz - 1 ;                                                    00745500
   end;                                                                      00745600
   underflo := false;                                                        00745700
   repeat begin                                                              00745800
{                                                                            00745900
      loop to construct xmin                                                 00746000
      exit from loop is signalled by an underflow                            00746100
                                                                          } 00746200
      xmin := y ;                                                            00746300
      y := y * betain ;                                                      00746400
      if ( ( ( y * one ) = zero ) or ( abs ( y )> xmin ) )                   00746500
      then begin                                                             00746600
         underflo := true;                                                   00746700
      end else begin                                                         00746800
         k := k + 1 ;                                                        00746900
      end;                                                                   00747000
   end until underflo ;                                                      00747100
   minexp := - k ;                                                           00747200
{  determine maxexp, xmax                                                    00747300
                                                                          } 00747400
   if ( ( mx <= k + k - 3 ) and ( ibeta <> 10 ) ) then begin                 00747500
      mx := mx + mx ;                                                        00747600
      iexp := iexp + 1 ;                                                     00747700
   end;                                                                      00747800
   maxexp := mx + minexp ;                                                   00747900
{  adjust for machines with implicit leading                                 00748000
   bit in binary significand and machines with                               00748100
   radix point at extreme right of significand                               00748200
                                                                          } 00748300
   i := maxexp + minexp ;                                                    00748400
   if ( ( ibeta = 2 ) and ( i = 0 ) ) then maxexp := maxexp - 1 ;            00748500
   if ( i > 20 ) then maxexp := maxexp - 3 ;                                 00748600
   xmax := one - epsneg ;                                                    00748700
   if ( xmax * one <> xmax ) then xmax := one - beta * epsneg ;              00748800
   xmax := ( xmax * betain * betain * betain ) / xmin ;                      00748900
   i := maxexp + minexp + 3 ;                                                00749000
   if  ( i > 0 ) then begin                                                  00749100
                                                                             00749200
      for j := 1 to i do begin                                               00749300
         xmax := xmax * beta ;                                               00749400
      end ;                                                                  00749500
   end;                                                                      00749600
                                                                             00749700
end;                                                                         00749800
                                                                             00749900
                                                                             00750000
```

```
                                                                             00750100
begin                                                                        00750200
   machar ( ibeta , it , irnd , ngrd , machep , negep , iexp , minexp ,     00750300
      maxexp , eps , epsneg , xmin , xmax );                                 00750400
   writeln(' OUTPUT FROM MACHAR');                                           00750500
   writeln;                                                                  00750600
   writeln;                                                                  00750700
   writeln(' BETA =',ibeta:5);                                               00750800
   writeln;                                                                  00750900
   writeln('    T =',it:5);                                                  00751000
   writeln;                                                                  00751100
   writeln('  RND =',irnd:5);                                                00751200
   writeln;                                                                  00751300
   writeln(' NGRD =',ngrd:5);                                                00751400
   writeln;                                                                  00751500
   writeln(' MACHEP =',machep:5);                                            00751600
   writeln;                                                                  00751700
   writeln(' NEGEP =',negep:5);                                              00751800
   writeln;                                                                  00751900
   writeln(' IEXP =',iexp:5);                                                00752000
   writeln;                                                                  00752100
   writeln(' MINEXP =',minexp:5);                                            00752200
   writeln;                                                                  00752300
   writeln(' MAXEXP =',maxexp:5);                                            00752400
   writeln;                                                                  00752500
   writeln(' EPS =',eps:15);                                                 00752600
   writeln;                                                                  00752700
   writeln(' EPSNEG =',epsneg:15);                                           00752800
   writeln;                                                                  00752900
   writeln(' XMIN =',xmin:15);                                               00753000
   writeln;                                                                  00753100
   writeln(' XMAX =',xmax:15);                                               00753200
end.                                                                         00753300
```

```
{TEST 6.6.6.3-1, CLASS=CONFORMANCE}

{ This program checks the implementation of the transfer
  functions trunc and round.
  The compiler fails if the program does not compile and run. }

program t6p6p6p3d1(output);
var
   i,
   truncstatus,
   roundstatus : integer;
   j : real;
begin
   truncstatus:=0;
   roundstatus:=0;
   if (trunc(3.7)=3) and (trunc(-3.7)=-3) then
      truncstatus:=truncstatus+1
   else
      writeln(' FAIL...6.6.6.3-1 : TRUNC');

   if (round(3.7)=4) and (round(-3.7)=-4) then
      roundstatus:=roundstatus+1
   else
      writeln(' FAIL...6.6.6.3-1 : ROUND');

   j:=0;
   for i:=-333 to 333 do
   begin
      j:=j+i div 100;
      if j<0 then
         if (trunc(j-0.5)=round(j)) then
            begin
               truncstatus:=truncstatus+1;
               roundstatus:=roundstatus+1
            end
         else
            writeln(' FAIL...6.6.6.3-1 : TRUNC ROUND')
      else
         if (trunc(j+0.5)=round(j)) then
            begin
               truncstatus:=truncstatus+1;
               roundstatus:=roundstatus+1
            end
         else
            writeln(' FAIL...6.6.6.3-1 : TRUNC ROUND')
   end;

   if (truncstatus=668) and (roundstatus=668) then
      writeln(' PASS...6.6.6.3-1')
end.
```

```
{TEST 6.6.6.3-2, CLASS=ERRORHANDLING}

{ This program causes an error to occur as the result
  returned by the trunc function is not a value of the
  type integer.
  The error should be detected at run-time. }

program t6p6p6p3d2(output);
var
   reel : real;
   i    : integer;
   ok   : boolean;
begin
   reel:=11111.11111;
   ok:=true;
   while ok do
   begin
      i:=trunc(reel);
      if (i<0) then
         ok:=false
      else
         reel:=reel*2
   end;
   writeln(' ERROR NOT DETECTED...6.6.6.3-2')
end.
```

```
{TEST 6.6.6.3-3, CLASS=ERRORHANDLING}

{ This program causes an error to occur as the result
  returned by the round function is not a value of the
  type integer.
  The error should be detected at run-time. }

program t6p6p6p3d3(output);
var
   reel : real;
   i    : integer;
   ok   : boolean;
begin
   reel:=11111.11111;
   ok:=true;
   while ok do
   begin
      i:=round(reel);
      if (i<0) then
         ok:=false
      else
         reel:=reel*2
   end;
   writeln(' ERROR NOT DETECTED...6.6.6.3-3')
end.
```

```
{TEST 6.6.6.3-4, CLASS=DEVIANCE}                                        00763400
                                                                        00763500
{ This test checks that neither trunc nor round are permitted           00763600
  to have integer parameters. The Standard requires these to be         00763700
  real. The compiler deviates if the program compiles and prints        00763800
  DEVIATES. }                                                           00763900
                                                                        00764000
program t6p6p6p3d4(output);                                             00764100
var                                                                     00764200
   i:integer;                                                           00764300
   x:real;                                                              00764400
begin                                                                   00764500
   i:=1979;                                                             00764600
   x:=trunc(i)+round(i+1);                                              00764700
   writeln(' DEVIATES...6.6.6.3-4, TRUNC/ROUND')                        00764800
end.                                                                    00764900
```

```
{TEST 6.6.6.4-1, CLASS=CONFORMANCE}                                     00765000
                                                                        00765100
{ This program checks that the implementation of the ord                00765200
  function is as described by the Standard.                             00765300
  The compiler fails if the program does not compile and run. }         00765400
                                                                        00765500
program t6p6p6p4d1(output);                                             00765600
type                                                                    00765700
   colourtype = (red,orange,yellow,green,blue);                         00765800
var                                                                     00765900
   colour   : colourtype;                                               00766000
   some     : orange..green;                                            00766100
   i        : integer;                                                  00766200
   counter  : integer;                                                  00766300
   ok       : boolean;                                                  00766400
begin                                                                   00766500
   counter:=0;                                                          00766600
   if (ord(false)=0) and (ord(true)=1) then                             00766700
      counter:=counter+1                                                00766800
   else                                                                 00766900
      writeln(' FAIL...6.6.6.4-1 : FALSE/TRUE');                        00767000
                                                                        00767100
   if (ord(red)=0) and (ord(orange)=1) and                              00767200
      (ord(yellow)=2) and (ord(green)=3) and                            00767300
      (ord(blue)=4) then                                                00767400
      counter:=counter+1                                                00767500
   else                                                                 00767600
      writeln(' FAIL...6.6.6.4-1 : COLOURTYPE');                        00767700
                                                                        00767800
   i:=-11;                                                              00767900
   ok:=true;                                                            00768000
   while ok do                                                          00768100
   begin                                                                00768200
      i:=i+1;                                                           00768300
      if i>10 then                                                      00768400
         ok:=false                                                      00768500
      else                                                              00768600
         if ord(i)=i then                                               00768700
            counter:=counter+1                                          00768800
         else                                                           00768900
         begin                                                          00769000
            ok:=false;                                                  00769100
            writeln(' FAIL...6.6.6.4-1 : I')                            00769200
         end                                                            00769300
   end;                                                                 00769400
                                                                        00769500
   colour:=blue;                                                        00769600
   some:=orange;                                                        00769700
   if ord(colour)=4 then                                                00769800
      counter:=counter+1                                                00769900
   else                                                                 00770000
      writeln(' FAIL...6.6.6.4-1 : COLOUR');                            00770100
                                                                        00770200
   if ord(some)=1 then                                                  00770300
      counter:=counter+1                                                00770400
   else                                                                 00770500
      writeln(' FAIL...6.6.6.4-1 : SOME');                              00770600
                                                                        00770700
   if counter=25 then                                                   00770800
      writeln(' PASS...6.6.6.4-1')                                      00770900
end.                                                                    00771000
```

```
{TEST 6.6.6.4-2, CLASS=CONFORMANCE}

{ This program checks the implementation of chr.
  The compiler fails if the program does not compile and run. }

program t6p6p6p4d2(output);
var
   letter : char;
   counter : integer;
begin
   counter:=0;

   for letter:='0' to '9' do
      if chr(ord(letter))=letter then
         counter:=counter+1;

   if counter=10 then
      writeln(' PASS...6.6.6.4-2')
   else
      writeln(' FAIL...6.6.6.4-2')
end.


{TEST 6.6.6.4-3, CLASS=CONFORMANCE}

{ This program tests the function pred only. The user is
  refered to tests 6.4.2.2-4 and 6.4.2.3-2 for tests of
  succ.
  The compiler fails if the program does not compile and run. }

program t6p6p6p4d3(output);
type
   colourtype = (red,orange,yellow,green,blue);
var
   colour : colourtype;
   counter: integer;
begin
   counter:=0;
   colour:=blue;
   colour:=pred(colour);
   colour:=pred(colour);
   colour:=pred(succ(colour));
   if colour=yellow then
      counter:=1
   else
      writeln(' FAIL...6.6.6.4-3 : COLOUR');

   if pred(-10)=-11 then
      counter:=counter+1
   else
      writeln(' FAIL...6.6.6.4-3 : -VE NUMBERS');

   if counter=2 then
      writeln(' PASS...6.6.6.4-3')
end.
```

```
{TEST 6.6.6.4-4, CLASS=ERRORHANDLING}

{ This program causes an error to occur as the
  function succ is applied to the last value
  of an ordinal type.
  The error should be detected by the compiler
  or at run-time. }

program t6p6p6p4d4(output);
type
   enumerated = (first,second,third,last);
var
   ordinal : enumerated;
begin
   ordinal:=succ(last);
   writeln(' ERROR NOT DETECTED...6.6.6.4-4')
end.


{TEST 6.6.6.4-5, CLASS=ERRORHANDLING}

{ This program causes an error to occur as the function PRED
  is applied to the first value of an ordinal type.
  The error should be detected by the compiler or at run-time. }

program t6p6p6p4d5(output);
type
   enumerated = (first,second,third,fourth,last);
var
   ordinal : enumerated;
begin
   ordinal:=first;
   ordinal:=pred(ordinal);
   writeln(' ERROR NOT DETECTED...6.6.6.4-5, PRED')
end.


{TEST 6.6.6.4-6, CLASS=DEVIANCE}

{ This test checks that succ and pred cannot be applied to
  real values. The compiler deviates if the program compiler
  and prints DEVIATES. }

program t6p6p6p4d6(output);
var
   x:real;
begin
   x:=0.3;
   if (succ(x)>x) and (pred(x)<x) then
      writeln(' DEVIATES...6.6.6.4-6, REAL SUCC/PRED')
   else
      writeln(' DEVIATES...6.6.6.4-6, MESS')
end.
```

```
{TEST 6.6.6.4-7, CLASS=ERRORHANDLING}                                   00781300
                                                                        00781400
{ This test evokes an error by pushing chr past the                     00781500
  limits of the char type. It assumes that no char type has more        00781600
  than 10000+ord('0') values. }                                         00781700
                                                                        00781800
program t6p6p6p4d7(output);                                             00781900
var                                                                     00782000
   i:0..10000;                                                          00782100
   c:char;                                                              00782200
begin                                                                   00782300
   for i:=0 to 10000 do                                                 00782400
      c:=chr(i+ord('0'));                                               00782500
   writeln(' ERROR NOT DETECTED...6.6.6.4-7, CHR')                      00782600
end.                                                                    00782700


{TEST 6.6.6.5-1, CLASS=CONFORMANCE}                                     00782800
                                                                        00782900
{ This program would have tested the function of the eof and eoln       00783000
  predicates. However, the test is carried out elsewhere, and           00783100
  the user is refered to tests 6.4.3.5-2 and 6.4.3.5-3. }               00783200
                                                                        00783300
program t6p6p6p5d1;                                                     00783400
begin                                                                   00783500
end.                                                                    00783600


{TEST 6.6.6.5-2, CLASS=CONFORMANCE}                                     00783700
                                                                        00783800
{ This program tests the predicate odd.                                 00783900
  The compiler fails if the program does not compile or                 00784000
  the program states that this is so. }                                 00784100
                                                                        00784200
program t6p6p6p5d2(output);                                             00784300
var                                                                     00784400
   i,counter : integer;                                                 00784500
function myodd(i:integer):boolean;                                      00784600
   begin                                                                00784700
      myodd := (abs(i mod 2) = 1);                                      00784800
   end;                                                                 00784900
begin                                                                   00785000
   counter:=0;                                                          00785100
   for i:=-10 to 10 do                                                  00785200
      if odd(i) then                                                    00785300
      begin                                                             00785400
         if myodd(i) then counter := counter+1                          00785500
      end else begin                                                    00785600
         if not myodd(i) then counter := counter+1                      00785700
      end;                                                              00785800
   if counter=21 then                                                   00785900
      writeln(' PASS...6.6.6.5-2')                                      00786000
   else                                                                 00786100
      writeln(' FAIL...6.6.6.5-2')                                      00786200
end.                                                                    00786300
```

```
{TEST 6.6.6.5-3, CLASS=DEVIANCE}                                        00786400
                                                                        00786500
{ This test checks that the function odd is restricted to               00786600
  integer parameters. This compiler deviates if the program compiles    00786700
  and prints DEVIATES. }                                                00786800
                                                                        00786900
program t6p6p6p5d3(output);                                             00787000
var                                                                     00787100
   x:real;                                                              00787200
begin                                                                   00787300
   x:=1.0;                                                              00787400
   if odd(x) then                                                       00787500
      writeln(' DEVIATES...6.6.6.5-3, REAL ODD')                        00787600
   else                                                                 00787700
      writeln(' DEVIATES...6.6.6.5-3, MESS')                            00787800
end.                                                                    00787900


{TEST 6.7.1-1, CLASS=CONFORMANCE}                                       00788000
                                                                        00788100
{ This program tests that the precedence of the boolean operators       00788200
  is as described in the Pascal Standard.                               00788300
  The compiler fails if the program does not compile, or the            00788400
  program states that this is so. }                                     00788500
                                                                        00788600
program t6p7pld1(output);                                               00788700
var                                                                     00788800
   a, b, c, w, x : boolean;                                             00788900
   counter:integer;                                                     00789000
begin                                                                   00789100
  counter := 0;                                                         00789200
   for a := false to true do                                            00789300
      for b:= false to true do                                          00789400
         for c := false to true do                                      00789500
         begin                                                          00789600
            w := (a and b) < c;                                         00789700
            x := c > b and a;                                           00789800
            if (w=x) then counter := counter+1;                         00789900
         end;                                                           00790000
   if (counter=8) then                                                  00790100
      writeln(' PASS...6.7.1-1')                                        00790200
   else                                                                 00790300
      writeln(' FAIL...6.7.1-1')                                        00790400
end.                                                                    00790500
```

```
{TEST 6.7.1-2, CLASS=CONFORMANCE}                                    00790600
                                                                     00790700
{ This program tests that the precedence of the arithmetic           00790800
  operators is as described by the Pascal Standard.                  00790900
  The compiler fails if the program does not compile, or             00791000
  the program states that this is so. }                              00791100
                                                                     00791200
program t6p7p1d2(output);                                            00791300
var                                                                  00791400
   a,b,c,d,e,f,g : integer;                                          00791500
   h,i,j,k,l,m,n : real;                                             00791600
begin                                                                00791700
   a:=1;                                                             00791800
   b:=2;                                                             00791900
   c:=3;                                                             00792000
   d:=4;                                                             00792100
   e:=5;                                                             00792200
   f:=a-b+c-d;                                                       00792300
   g:=e-d div b*c;                                                   00792400
   h:=1;                                                             00792500
   i:=2;                                                             00792600
   j:=3;                                                             00792700
   k:=4;                                                             00792800
   l:=5;                                                             00792900
   m:=h/i*j/k;                                                       00793000
   n:=l+k/i-3*j;                                                     00793100
   if (f=-2) and (g=-1) and (n=-2) and                               00793200
      ((m<0.38) and (m>0.37)) then                                   00793300
      writeln(' PASS...6.7.1-2')                                     00793400
   else                                                              00793500
      writeln(' FAIL...6.7.1-2')                                     00793600
end.                                                                 00793700


{TEST 6.7.2.2-1, CLASS=CONFORMANCE}                                  00793800
                                                                     00793900
{ This program checks the operation of the operators + - and *.      00794000
  The compiler fails if the program does not compile, or the         00794100
  program states that this is so. }                                  00794200
                                                                     00794300
program t6p7p2p2d1(output);                                          00794400
var                                                                  00794500
   i, x, y , counter : integer;                                      00794600
begin                                                                00794700
   counter := 0;                                                     00794800
   for x := -10 to 10 do                                             00794900
   begin                                                             00795000
      if (succ(x)=x+1) then                                          00795100
         counter := counter+1;                                       00795200
      if (pred(x) = x-1) then                                        00795300
         counter := counter+1;                                       00795400
      if (x*x=sqr(x)) then                                           00795500
         counter:= counter+1;                                        00795600
   end;                                                              00795700
   if (counter=63) then                                              00795800
      writeln(' PASS...6.7.2.2-1')                                   00795900
   else                                                              00796000
      writeln(' FAIL...6.7.2.2-1')                                   00796100
end.                                                                 00796200
```

```
{TEST 6.7.2.2-2, CLASS=CONFORMANCE}                                  00796300
                                                                     00796400
{ This program checks that DIV and MOD are implemented by the        00796500
  rule specified by the Pascal Standard.                             00796600
  The compiler fails if the program does not compile, or the         00796700
  program states that this is so. }                                  00796800
                                                                     00796900
program t6p7p2p2d2(output);                                          00797000
var                                                                  00797100
   i, j, counter : integer;                                          00797200
begin                                                                00797300
   counter:=0;                                                       00797400
   for i:=0 to 6 do                                                  00797500
      for j:=1 to 4 do                                               00797600
         if ((i-j)<((i div j)*j)) and (((i div j)*j)<=i) then        00797700
            counter:=counter+1;                                      00797800
   for i:=0 to 6 do                                                  00797900
      for j:=1 to 4 do                                               00798000
         if (i mod j)=(i-(i div j)*j) then                           00798100
            counter:=counter+1;                                      00798200
   if counter=56 then                                                00798300
      writeln(' PASS...6.7.2.2-2')                                   00798400
   else                                                              00798500
      writeln(' FAIL...6.7.2.2-2: DIV MOD')                          00798600
end.                                                                 00798700


{TEST 6.7.2.2-3, CLASS=ERRORHANDLING}                                00798800
                                                                     00798900
{ This program causes an error to occur as the second operand        00799000
  of the DIV operator is 0.                                          00799100
  The error should be detected at run-time. }                        00799200
                                                                     00799300
program t6p7p2p2d3(output);                                          00799400
var                                                                  00799500
   i, j, k : integer;                                                00799600
begin                                                                00799700
   i:=6;                                                             00799800
   j:=0;                                                             00799900
   k:=i div j;        { an error as j=0 }                            00800000
   writeln(' ERROR NOT DETECTED...6.7.2.2-3: ZERO DIVIDE (DIV)')     00800100
end.                                                                 00800200
```

```
{TEST 6.7.2.2-4, CLASS=QUALITY}                                          00800300
                                                                         00800400
{ This program checks that constant and variable operands for DIV        00800500
  produce the same result, and if negative operands are permitted. }     00800600
                                                                         00800700
program t6p7p2p2d4(output);                                              00800800
var                                                                      00800900
   i, j, k, l, m,counter : integer;                                      00801000
begin                                                                    00801100
   { The next few statements may cause a run-time error. }               00801200
                                                                         00801300
   writeln(' THIS PROGRAM ATTEMPTS DIVISION WITH NEGATIVE OPERANDS');    00801400
   counter := 0;                                                         00801500
   j:=2;                                                                 00801600
   for i:= -10 to 10 do                                                  00801700
   begin                                                                 00801800
      l:=i div j;                                                        00801900
      m:= i div 2;                                                       00802000
      if (l=m) then counter := counter+1;                                00802100
      l:=i mod j;                                                        00802200
      m:= i mod 2;                                                       00802300
      if (l=m) then counter := counter+1;                                00802400
      if (i-i div 2 * 2 = i mod 2) then counter := counter+1;            00802500
   end;                                                                  00802600
   if counter = 63 then                                                  00802700
   begin                                                                 00802800
      write(' DIVISION INTO NEGATIVE OPERANDS IMPLEMENTED AND ');        00802900
      writeln('CONSISTENT');                                             00803000
   end else                                                              00803100
      writeln(' INCONSISTENT DIVISION INTO NEGATIVE OPERANDS');          00803200
   counter := 0;                                                         00803300
   j:=-2;                                                                00803400
   for i:= -10 to 10 do                                                  00803500
   begin                                                                 00803600
      l:=i div j;                                                        00803700
      m:= i div (-2);                                                    00803800
      if (l=m) then counter := counter+1;                                00803900
      l:=i mod j;                                                        00804000
      m:= i mod (-2);                                                    00804100
      if (l=m) then counter := counter+1;                                00804200
   end;                                                                  00804300
   if counter = 42 then                                                  00804400
   begin                                                                 00804500
      write(' DIVISION BY NEGATIVE OPERANDS IMPLEMENTED AND ');          00804600
      writeln('CONSISTENT');                                             00804700
   end else                                                              00804800
      writeln(' INCONSISTENT DIVISION BY NEGATIVE OPERANDS');            00804900
   i:=-3;                                                                00805000
   if (i div 2 = -1) then                                                00805100
      writeln(' QUOTIENT = TRUNC(A/B) FOR NEGATIVE OPERANDS')            00805200
   else                                                                  00805300
      writeln(' QUOTIENT = TRUNC(A/B-1) FOR NEGATIVE OPERANDS');         00805400
   if (i mod 2 = 1) then                                                 00805500
      writeln(' MOD(A,B) LIES IN (0,B-1)')                               00805600
   else                                                                  00805700
      writeln(' MOD RETURNS REMAINDER OF DIV');                          00805800
end.                                                                     00805900
```

```
{TEST 6.7.2.2-5, CLASS=CONFORMANCE}                                      00806000
                                                                         00806100
{ This program checks that maxint satisfies the conditions laid          00806200
  down in the Pascal Standard.                                           00806300
  The compiler fails if the program does not compile, or does            00806400
  not print pass. }                                                      00806500
                                                                         00806600
program t6p7p2p2d5(output);                                              00806700
var                                                                      00806800
   i : integer;                                                          00806900
begin                                                                    00807000
   i:=-(-maxint);                                                        00807100
   i:=-maxint;                                                           00807200
   if odd(maxint) then                                                   00807300
      i:=(maxint-((maxint div 2)+1))*2                                   00807400
   else                                                                  00807500
      i:=(maxint-(maxint div 2))*2;                                      00807600
   if i<=maxint then                                                     00807700
      writeln(' PASS...6.7.2.2-5')                                       00807800
   else                                                                  00807900
      writeln(' FAIL...6.7.2.2-5: MAXINT')                               00808000
end.                                                                     00808100


{TEST 6.7.2.2-6, CLASS=ERRORHANDLING}                                    00808200
                                                                         00808300
{ This program causes an error to occur as the result of a binary        00808400
  integer operation is not in the interval 0 -> +maxint. }               00808500
                                                                         00808600
program t6p7p2p2d6(output);                                              00808700
var                                                                      00808800
   i : integer;                                                          00808900
begin                                                                    00809000
   i:=(maxint-(maxint div 2))*2+2;                                       00809100
   writeln(' ERROR NOT DETECTED...6.7.2.2-6')                            00809200
end.                                                                     00809300


{TEST 6.7.2.2-7, CLASS=ERRORHANDLING}                                    00809400
                                                                         00809500
{ This program causes an error to occur as the result of a binary        00809600
  integer operation is not in the interval 0 -> -maxint. }               00809700
                                                                         00809800
program t6p7p2p2d7(output);                                              00809900
var                                                                      00810000
   i : integer;                                                          00810100
begin                                                                    00810200
   i:=(-maxint+(maxint div 2))*2-2;                                      00810300
   writeln(' ERROR NOT DETECTED...6.7.2.2-7')                            00810400
end.                                                                     00810500
```

```
{TEST 6.7.2.2-8, CLASS=ERRORHANDLING}                                   00810600
                                                                        00810700
{ This program causes an error to occur as the second operand           00810800
  of the MOD operator is 0.                                             00810900
  The error should be detected at run-time. }                           00811000
                                                                        00811100
program t6p7p2p2d8(output);                                             00811200
var                                                                     00811300
   i, j, k : integer;                                                   00811400
begin                                                                   00811500
   i:=6;                                                                00811600
   j:=0;                                                                00811700
   k:=i mod j;        { an error as j=0 }                               00811800
   writeln(' ERROR NOT DETECTED...6.7.2.2-8: MOD ZERO')                 00811900
end.                                                                    00812000


{TEST 6.7.2.2-9, CLASS=DEVIANCE}                                        00812100
                                                                        00812200
{ The unary operator plus can clearly only be applied to numeric        00812300
  operands. Hence this program should fail to compile.                  00812400
  The compiler deviates if the program compiles and prints              00812500
  DEVIATES. }                                                           00812600
                                                                        00812700
program t6p7p2p2d9(output);                                             00812800
const                                                                   00812900
   capa = 'A';                                                          00813000
begin                                                                   00813100
   writeln(+capa);                                                      00813200
   writeln(' DEVIATES...6.7.2.2-9, UNARY OPERATOR')                     00813300
end.                                                                    00813400
```

```
{TEST 6.7.2.3-1, CLASS=CONFORMANCE}                                     00813500
                                                                        00813600
{ This test checks the operation of the boolean operators.              00813700
  The compiler fails if the program does not compile, or the            00813800
  program states that this is so. }                                     00813900
                                                                        00814000
program t6p7p2p3d1(output);                                             00814100
var                                                                     00814200
   a,b,c : boolean;                                                     00814300
   counter : integer;                                                   00814400
begin                                                                   00814500
   counter:=0;                                                          00814600
   a:=false;                                                            00814700
   b:=false;                                                            00814800
   { OR truth table }                                                   00814900
                                                                        00815000
   if a or b then                                                       00815100
      writeln(' FAIL...6.7.2.3-1: OR')                                  00815200
   else                                                                 00815300
   begin                                                                00815400
      b:=true;                                                          00815500
      if a or b then                                                    00815600
      begin                                                             00815700
         a:=true;                                                       00815800
         b:=false;                                                      00815900
         if a or b then                                                 00816000
         begin                                                          00816100
            b:=true;                                                    00816200
            if a or b then                                              00816300
               counter:=counter+1                                       00816400
            else                                                        00816500
               writeln(' FAIL...6.7.2.3-1: OR')                         00816600
         end                                                            00816700
         else                                                           00816800
            writeln(' FAIL...6.7.2.3-1: OR')                            00816900
      end                                                               00817000
      else                                                              00817100
         writeln(' FAIL...6.7.2.3-1: OR')                               00817200
   end;                                                                 00817300
                                                                        00817400
   { AND truth table }                                                  00817500
   a:=false;                                                            00817600
   b:=false;                                                            00817700
   if a and b then                                                      00817800
      writeln(' FAIL...6.7.2.3-1: AND')                                 00817900
   else                                                                 00818000
   begin                                                                00818100
      b:=true;                                                          00818200
      if a and b then                                                   00818300
         writeln(' FAIL...6.7.2.3-1: AND')                              00818400
      else                                                              00818500
      begin                                                             00818600
         a:=true;                                                       00818700
         b:=false;                                                      00818800
         if a and b then                                                00818900
            writeln(' FAIL...6.7.2.3-1: AND')                           00819000
         else                                                           00819100
         begin                                                          00819200
            b:=true;                                                    00819300
            if a and b then                                             00819400
               counter:=counter+1                                       00819500
```

```
         else                                                          00819600
            writeln(' FAIL...6.7.2.3-1: AND')                          00819700
       end                                                             00819800
    end                                                                00819900
end;                                                                   00820000
                                                                       00820100
{ NOTE: NOT is sometimes badly implemented by wordwise                 00820200
        complementation, and for this reason the following             00820300
        two tests may fail. }                                          00820400
                                                                       00820500
if (not false)=true then                                               00820600
   counter:=counter+1                                                  00820700
else                                                                   00820800
   writeln(' FAIL...6.7.2.3-1: NOT FALSE');                            00820900
                                                                       00821000
if (not true)=false then                                               00821100
   counter:=counter+1                                                  00821200
else                                                                   00821300
   writeln(' FAIL...6.7.2.3-1: NOT TRUE');                             00821400
                                                                       00821500
c:=false;                                                              00821600
a:=true;                                                               00821700
b:=false;                                                              00821800
if (a or b)=(b or a) then                                              00821900
   counter:=counter+1                                                  00822000
else                                                                   00822100
   writeln(' FAIL...6.7.2.3-1: COMMUTATION');                          00822200
                                                                       00822300
if ((a or b)or c)=(a or(b or c)) then                                  00822400
   counter:=counter+1                                                  00822500
else                                                                   00822600
   writeln(' FAIL...6.7.2.3-1: ASSOCIATIVITY');                        00822700
                                                                       00822800
if (a and(b or c))=((a and b)or(a and c)) then                         00822900
   counter:=counter+1                                                  00823000
else                                                                   00823100
   writeln(' FAIL...6.7.2.3-1: DISTRIBUTION');                         00823200
                                                                       00823300
if not(a or b)=((not a) and(not b)) then                               00823400
   counter:=counter+1                                                  00823500
else                                                                   00823600
   writeln(' FAIL...6.7.2.3-1: DEMORGAN1');                            00823700
                                                                       00823800
if not(a and b)=((not a) or (not b)) then                              00823900
   counter:=counter+1                                                  00824000
else                                                                   00824100
   writeln(' FAIL...6.7.2.3-1: DEMORGAN2');                            00824200
                                                                       00824300
if not(not a)= a then                                                  00824400
   counter:=counter+1                                                  00824500
else                                                                   00824600
   writeln(' FAIL...6.7.2.3-1: INVERSION');                            00824700
                                                                       00824800
if counter=10 then                                                     00824900
   writeln(' PASS...6.7.2.3-1')                                        00825000
end.                                                                   00825100
```

```
{TEST 6.7.2.3-2, CLASS=IMPLEMENTATIONDEFINED}                          00825200
                                                                       00825300
{ This program determines if a boolean expression is partially         00825400
  evaluated if the value of the expression is determined before        00825500
  the expression is fully evaluated }                                  00825600
                                                                       00825700
program t6p7p2p3d2(output);                                            00825800
var                                                                    00825900
   a:boolean;                                                          00826000
   k,l:integer;                                                        00826100
function sideeffect(var i:integer; b:boolean):boolean;                 00826200
begin                                                                  00826300
   i:=i+1;                                                             00826400
   sideeffect:=b;                                                      00826500
end;                                                                   00826600
                                                                       00826700
begin                                                                  00826800
   writeln(' TEST OF SHORT CIRCUIT EVALUATION OF (A AND B)');          00826900
   k:=0;                                                               00827000
   l:=0;                                                               00827100
   a:=sideeffect(k,false) and sideeffect(l,false);                     00827200
   if (k=0) and (l=1) then                                             00827300
      writeln(' SECOND EXPRESSION EVALUATED...6.7.2.3-2')              00827400
   else                                                                00827500
      if (k=1) and (l=0) then                                          00827600
         writeln(' FIRST EXPRESSION EVALUATED...6.7.2.3-2')            00827700
      else                                                             00827800
         if(k=1) and (l=1) then                                        00827900
            writeln(' BOTH EXPRESSIONS EVALUATED...6.7.2.3-2')         00828000
         else                                                          00828100
            writeln(' FAIL...6.7.2.3-2');                              00828200
end.                                                                   00828300
```

```
{TEST 6.7.2.3-3, CLASS=IMPLEMENTATIONDEFINED}                                00828400
                                                                             00828500
{ This program determines if a boolean expression is partially               00828600
  evaluated if the value of the expression is determined before              00828700
  the expression is fully evaluated }                                        00828800
                                                                             00828900
program t6p7p2p3d3(output);                                                  00829000
var                                                                          00829100
   a:boolean;                                                                00829200
   k,l:integer;                                                              00829300
                                                                             00829400
function sideeffect(var i:integer; b:boolean):boolean;                       00829500
begin                                                                        00829600
   i:=i+1;                                                                   00829700
   sideeffect:=b;                                                            00829800
end;                                                                         00829900
                                                                             00830000
begin                                                                        00830100
   writeln(' TEST OF SHORT CIRCUIT EVALUATION OF (A OR B)');                 00830200
   k:=0;                                                                     00830300
   l:=0;                                                                     00830400
   a:=sideeffect(k,true) or sideeffect(l,true);                              00830500
   if (k=0) and (l=1) then                                                   00830600
      writeln(' SECOND EXPRESSION EVALUATED...6.7.2.3-3')                    00830700
   else                                                                      00830800
      if (k=1) and (l=0) then                                                00830900
         writeln(' FIRST EXPRESSION EVALUATED...6.7.2.3-3')                  00831000
      else                                                                   00831100
         if(k=1) and (l=1) then                                              00831200
            writeln(' BOTH EXPRESSIONS EVALUATED...6.7.2.3-3')               00831300
         else                                                                00831400
            writeln(' FAIL...6.7.2.3-3');                                    00831500
end.                                                                         00831600


{TEST 6.7.2.3-4, CLASS=DEVIANCE}                                             00831700
                                                                             00831800
{ Are logical operators allowed  to be performed on integers?                00831900
  The compiler deviates if the program compiles and prints                   00832000
  DEVIATES. }                                                                00832100
                                                                             00832200
program t6p7p2p3d4(output);                                                  00832300
var                                                                          00832400
  i,j:integer;                                                               00832500
begin                                                                        00832600
   i:=1; j:=2;                                                               00832700
   i:=i and j;                                                               00832800
   i:=i and 1;                                                               00832900
   i:= i or j;                                                               00833000
   i:=i or j;                                                                00833100
   i:= not j;                                                                00833200
   writeln(' DEVIATES...6.7.2.3-4, LOGICAL OPS.')                            00833300
end.                                                                         00833400
```

```
{TEST 6.7.2.4-1, CLASS=ERRORHANDLING}                                        00833500
                                                                             00833600
{ This test checks that operations on overlapping sets are detected.         00833700
  An error should be detected by the compiler or produced                    00833800
  at run time. }                                                             00833900
                                                                             00834000
program t6p7p2p4d1(output);                                                  00834100
var                                                                          00834200
   a,d : set of 0..10;                                                       00834300
   b,c : set of 5..15;                                                       00834400
begin                                                                        00834500
   b:=[5,10];                                                                00834600
   a:=[0,5,10];                                                              00834700
   d:=a+b;   {ok}                                                            00834800
   b:=[5,10,15];                                                             00834900
   c:=a+b;   {should be an error}                                            00835000
   writeln(' ERROR NOT DETECTED...6.7.2.4-1: OVERLAPPING SETS');             00835100
end.                                                                         00835200


{TEST 6.7.2.4-2, CLASS=CONFORMANCE}                                          00835300
                                                                             00835400
{ This test checks the operation of set operators.                           00835500
   The compiler fails if the program does not compile, or the                00835600
  program states that this is so. }                                          00835700
                                                                             00835800
program t6p7p2p4d2(output);                                                  00835900
var                                                                          00836000
   a,b,c,d:set of 0..10;                                                     00836100
   counter:integer;                                                          00836200
begin                                                                        00836300
   counter :=0;                                                              00836400
   a:=[0,2,4,6,8,10];                                                        00836500
   b:=[1,3,5,7,9];                                                           00836600
   c:=[];                                                                    00836700
   d:=[0,1,2,3,4,5,6,7,8,9,10];                                              00836800
   if (a+b=d) then                                                           00836900
      counter:=counter+1;                                                    00837000
   if (d-b=a) then                                                           00837100
      counter := counter+1;                                                  00837200
   if (d*b=b) then                                                           00837300
      counter:=counter+1;                                                    00837400
   if(d*b-b=c) then                                                          00837500
      counter:=counter+1;                                                    00837600
   if (a+b+c=d) then                                                         00837700
      counter:=counter+1;                                                    00837800
   if(counter=5) then                                                        00837900
      writeln(' PASS...6.7.2.4-2, SET OPERATORS')                            00838000
   else                                                                      00838100
      writeln(' FAIL...6.7.2.4-2, SET OPERATORS');                           00838200
end.                                                                         00838300
```

```
{TEST 6.7.2.4-3, CLASS=CONFORMANCE}                                    00838400
                                                                       00838500
{ This program checks the operations of set operators on sets          00838600
  of constants and variables. The compiler fails if the program        00838700
  does not compile or the program states that this is so. }            00838800
                                                                       00838900
program t6p7p2p4d3(output);                                            00839000
var                                                                    00839100
   a,b,c:set of 0..10;                                                 00839200
   counter:integer;                                                    00839300
begin                                                                  00839400
   counter:=0;                                                         00839500
   a:=[0,2,4,6,8,10];                                                  00839600
   b:=[1,3,5,7,9];                                                     00839700
   c:=[0,1,2,3,4,5,6,7,8,9,10];                                        00839800
   if(a+[]=a) then                                                     00839900
      counter:=counter+1;                                              00840000
   if(a+b=c) then                                                      00840100
      counter:=counter+1;                                              00840200
   if(a+[1,3,5,7,9]=c) then                                            00840300
      counter:=counter+1;                                              00840400
   if(a-[]=a) then                                                     00840500
      counter:=counter+1;                                              00840600
   if(c-a=b) then                                                      00840700
      counter:=counter+1;                                              00840800
   if(c-[0,2,4,6,8,10]=b) then                                         00840900
      counter:=counter+1;                                              00841000
   if(a*a=a) then                                                      00841100
      counter:=counter+1;                                              00841200
   if(a*[]=[]) then                                                    00841300
      counter:=counter+1;                                              00841400
   if(a*b=[]) then                                                     00841500
      counter:=counter+1;                                              00841600
   if(a*c=a) then                                                      00841700
      counter:=counter+1;                                              00841800
   if(counter=10) then                                                 00841900
      writeln(' PASS...6.7.2.4-3 SET OPERATORS')                       00842000
   else                                                                00842100
      writeln('FAIL...6.7.2.4-3 SET OPERATORS');                       00842200
end.                                                                   00842300
```

```
{TEST 6.7.2.5-1, CLASS=CONFORMANCE}                                    00842400
                                                                       00842500
{ This program tests the use of relational operators on strings.       00842600
  The operators denote lexicographic ordering according to the         00842700
  ordering of the character set.                                       00842800
  The compiler fails if the program does not compile, or the           00842900
  program states that this is so. }                                    00843000
                                                                       00843100
program t6p7p2p5d1(output);                                            00843200
type                                                                   00843300
   string=packed array[1..7] of char;                                  00843400
var                                                                    00843500
   string1,                                                            00843600
   string2 : string;                                                   00843700
begin                                                                  00843800
   string1:='STRING1';                                                 00843900
   string2:='STRING2';                                                 00844000
   if (string1<>string2) and (string1<string2) then                    00844100
   begin                                                               00844200
      string1:='STRINGS';                                              00844300
      string2:='STRINGZ';                                              00844400
      if (string1<>string2) and (string1<string2) then                 00844500
         writeln(' PASS...6.7.2.5-1')                                  00844600
      else                                                             00844700
         writeln(' FAIL...6.7.2.5-1')                                  00844800
   end                                                                 00844900
   else                                                                00845000
      writeln(' FAIL...6.7.2.5-1')                                     00845100
end.                                                                   00845200
```

```
{TEST 6.7.2.5-2, CLASS=CONFORMANCE}                                    00845300
                                                                       00845400
{ This test checks the use of relational operators on sets.            00845500
  The compiler fails if the program does not compile, or the           00845600
  program states that this is so. }                                    00845700
                                                                       00845800
program t6p7p2p5d2(output);                                            00845900
var                                                                    00846000
   a,b:set of 0..10;                                                   00846100
  c,counter:integer;                                                   00846200
begin                                                                  00846300
   counter:=0;                                                         00846400
   a:=[0,1,2,3,4,5];                                                   00846500
   b:=[2,3,4];                                                         00846600
   c:=3;                                                               00846700
   if(a=[0,1,2,3,4,5]) then                                            00846800
     counter:=counter+1;                                               00846900
   if(a<>b) then                                                       00847000
      counter:=counter+1;                                              00847100
   if(b<>[1,2,3,4,5]) then                                             00847200
      counter:=counter+1;                                              00847300
   if(b<=a) then                                                       00847400
      counter:=counter+1;                                              00847500
   if(a>=b) then                                                       00847600
      counter:=counter+1;                                              00847700
   if([0,1]<=a) then                                                   00847800
      counter:=counter+1;                                              00847900
   if([1,2,3,4,5,6,10]>=b) then                                        00848000
      counter:=counter+1;                                              00848100
   if (1 in a) then                                                    00848200
      counter:=counter+1;                                              00848300
   if(c in b) then                                                     00848400
      counter:=counter+1;                                              00848500
   if(counter=9) then                                                  00848600
      writeln(' PASS...6.7.2.5-2 SET RELATIONAL OPERATORS')            00848700
   else                                                                00848800
      writeln(' FAIL...6.7.2.5-2 SET RELATIONAL OPERATORS');           00848900
end.                                                                   00849000
```

```
{TEST 6.7.2.5-3, CLASS=DEVIANCE}                                       00849100
                                                                       00849200
{ This test checks that file comparisons are not allowed.              00849300
  The semantics of this situation are particularly ill-defined,        00849400
  and not within standard Pascal. The compiler deviates if the         00849500
  program compiles and prints DEVIATES. }                              00849600
                                                                       00849700
program t6p7p2p5d3(output);                                            00849800
var                                                                    00849900
   f:text;                                                             00850000
begin                                                                  00850100
   rewrite(f);                                                         00850200
   if f=output then                                                    00850300
      writeln(' FAIL1...6.7.2.5-3, CONTENTS COMPARED')                 00850400
   else                                                                00850500
      writeln(' FAIL2...6.7.2.5-3, DESCRIPTORS COMPARED')              00850600
end.                                                                   00850700
```

```
{TEST 6.7.2.5-4, CLASS=DEVIANCE}                                       00850800
                                                                       00850900
{ Are relational operators permitted to concatenate?                   00851000
  The compiler deviates if the program compiles and                    00851100
  prints DEVIATES. }                                                   00851200
                                                                       00851300
program t6p7p2p5d4(output);                                            00851400
var                                                                    00851500
   x,y,z:integer;                                                      00851600
  b:boolean;                                                           00851700
begin                                                                  00851800
   x:=1;                                                               00851900
   y:=2;                                                               00852000
   z:=3;                                                               00852100
   b:=(x<y<z);                                                         00852200
   writeln(' DEVIATES...6.7.2.5-4, REL. OPS. ')                        00852300
end.                                                                   00852400
```

```
{TEST 6.8.2.1-1, CLASS=CONFORMANCE}                                    00852500
                                                                       00852600
{ Does the compiler allow all the possible empty clauses?              00852700
  The compiler fails if the program does not compile and print         00852800
  PASS. }                                                              00852900
                                                                       00853000
program t6p8p2p1d1(output);                                            00853100
var                                                                    00853200
   b:boolean;                                                          00853300
   r1:record                                                           00853400
       x:real;                                                         00853500
       a:integer;   {1}                                                00853600
       end;                                                            00853700
   r2:record                                                           00853800
       case b:boolean of                                               00853900
       true:(                                                          00854000
            c:real;                                                    00854100
            d:char;   {2}                                              00854200
           );                                                          00854300
       false:                                                          00854400
            (e:integer);   {3}                                         00854500
       end;                                                            00854600
begin                                                                  00854700
   b:=true;                                                            00854800
  if b then;   {4}                                                     00854900
   if b then else;   {5}                                               00855000
   repeat                                                              00855100
      b:= not b;   {6}                                                 00855200
   until b;                                                            00855300
   while b do                                                          00855400
   begin                                                               00855500
      b:=not b;   {7}                                                  00855600
   end;                                                                00855700
   with r1 do;   {8}                                                   00855800
  r1.a:=1;                                                             00855900
   case r1.a of                                                        00856000
   0:  b:=false;                                                       00856100
   1:  ;   {9}                                                        00856200
   2:  b:=true;   {10}                                                 00856300
   end;                                                                00856400
   writeln(' PASS...6.8.2.1-1, EMPTY STATEMENT');   {11}               00856500
end.                                                                   00856600
```

```
{TEST 6.8.2.2-1, CLASS=IMPLEMENTATIONDEFINED}                          00856700
                                                                       00856800
{ This program determines whether selection of a variable involving    00856900
  the indexing of an array occurs before or after the evaluation       00857000
  of the expression in an assignment statement. }                      00857100
                                                                       00857200
program t6p8p2p2d1(output);                                            00857300
var                                                                    00857400
   i : integer;                                                        00857500
   a : array[1..3] of integer;                                         00857600
function sideeffect(var i:integer) : integer;                          00857700
begin                                                                  00857800
   i:=i+1;                                                             00857900
   sideeffect:=i                                                       00858000
end;                                                                   00858100
                                                                       00858200
begin                                                                  00858300
   writeln(' TEST OF BINDING ORDER (A[I] := EXP)');                    00858400
   i:=1;                                                               00858500
   a[1]:=0;                                                            00858600
   a[2]:=0;                                                            00858700
   a[i]:=sideeffect(i);                                                00858800
   if a[1]=2 then                                                      00858900
      writeln(' SELECTION THEN EVALUATION...6.8.2.2-1')                00859000
   else                                                                00859100
      if a[2]=2 then                                                   00859200
         writeln(' EVALUATION THEN SELECTION...6.8.2.2-1')             00859300
end.                                                                   00859400
```

```
{TEST 6.8.2.2-2, CLASS=IMPLEMENTATIONDEFINED}

{ This program is similar to 6.8.2.2-1, except that the
  selection of the variable involves the dereferencing of
  a pointer. }

program t6p8p2p2d2(output);
type
   rekord=record
            a : integer;
            b : boolean;
            link : ↑rekord
          end;
   poynter=↑rekord;
var
   temp, ptr : poynter;
function sideeffect(var p : poynter) : integer;
begin
   p:=p↑.link;
   sideeffect:=2;
end;

begin
   writeln(' TEST OF BINDING ORDER (P↑ := EXP)');
   new(ptr);
   ptr↑.a:=1;
   ptr↑.b:=true;
   new(temp);
   ptr↑.link:=temp;
   temp↑.a:=0;
   temp↑.b:=false;
   temp:=ptr;
   ptr↑.a:=sideeffect(ptr);
   if temp↑.a=2 then
      writeln(' SELECTION THEN EVALUATION...6.8.2.2-2')
   else
      if temp↑.link↑.a=2 then
         writeln(' EVALUATION THEN SELECTION...6.8.2.2-2')
end.
```

```
{TEST 6.8.2.4-1, CLASS=CONFORMANCE}

{ This test checks that non-local goto statements are allowed }

program t6p8p2p4d1(output);
label 1;
var
   b:boolean;
procedure q;
begin
   b:=true;
   goto 1;
end;   {of q}

begin   {main}
   q;
   b:=false;
1: if b then
      writeln(' PASS...6.8.2.4-1 NON-LOCAL GOTO')
   else
      writeln(' FAIL...6.8.2.4-1 NON-LOCAL GOTO');
end.


{TEST 6.8.2.4-2, CLASS=DEVIANCE}

{ This test checks whether jumps between branches of an if
  statement are allowed.
  The compiler deviates if the program compiles and the
  program prints DEVIATES. }

program t6p8p2p4d2(output);
label
   1,2;
var
   i:integer;
begin
   i:=5;
   if (i<10) then
      goto 1
   else
      1:writeln(' DEVIATES...6.8.2.4-2');
   if (i>10) then
      2: writeln(' DEVIATES...6.8.2.4-2')
   else
     goto 2;
end.
```

```
{TEST 6.8.2.4-3, CLASS=DEVIATES}                                    00867900
                                                                    00868000
{ This test checks whether jumps between branches of a              00868100
  case statement are allowed. The compiler deviates if the          00868200
  program compiles and the program prints DEVIATES. }               00868300
                                                                    00868400
program t6p8p2p4d3(output);                                         00868500
label                                                               00868600
   4;                                                               00868700
var                                                                 00868800
   i:1..3;                                                          00868900
begin                                                               00869000
   for i:=1 to 2 do                                                 00869100
      case i of                                                     00869200
      1:  ;                                                         00869300
      2: goto 4;                                                    00869400
      3:4:                                                          00869500
         writeln(' DEVIATES...6.8.2.4-3');                          00869600
      end;                                                          00869700
end.                                                                00869800
```

```
{TEST 6.8.2.4-4,CLASS=DEVIANCE}                                     00869900
                                                                    00870000
{ This test checks that a goto statement causes an error            00870100
  when the statement(S) to which control is transferred is          00870200
  not activated either by S or a statement in the statement         00870300
  sequence of which S is an immediate constituent.                  00870400
  The compiler deviates if the compiler prints DEVIATES. }          00870500
                                                                    00870600
program t6p8p2p4d4(output);                                         00870700
var                                                                 00870800
   flag:boolean;                                                    00870900
                                                                    00871000
procedure a(i:integer;b:boolean);                                   00871100
label 99;                                                           00871200
   procedure r;                                                     00871300
   begin                                                            00871400
      goto 99;                                                      00871500
   end;                                                             00871600
begin                                                               00871700
   case i of                                                        00871800
   0:99: if (b) then                                                00871900
         writeln(' DEVIATES...6.8.2.4-4')                           00872000
      else                                                          00872100
         if flag then                                               00872200
            writeln(' PASS...6.8.2.4-4')                            00872300
         else begin                                                 00872400
            flag := true;                                           00872500
            a(1,false);                                             00872600
         end;                                                       00872700
   1:                                                               00872800
      a(2,true);                                                    00872900
   2:                                                               00873000
      r;                                                            00873100
   end;                                                             00873200
end;                                                                00873300
                                                                    00873400
begin                                                               00873500
   flag := false;                                                   00873600
   a(0,false);                                                      00873700
end.                                                                00873800
```

```
{TEST 6.8.3.4-1 CLASS=CONFORMANCE}

{ This test checks a nested if statement whose syntax is apparently
  ambiguous. The compiler fails if the program does not compile
  or the program states this by writing FAIL. }

program t6p8p3p4d1(output);
const
   off=false;
var
   b:boolean;
begin
   for b:=false to true do
      begin
      if b then
         if off then
            writeln(' FAIL...6.8.3.4-1')
         else
            begin
            if not b then
                writeln(' FAIL...6.8.3.4-1')
            else
               writeln(' PASS...6.8.3.4-1');
            end;
      end;
end.


{TEST 6.8.3.5-1, CLASS=CONFORMANCE}

{ This test checks that a minimal case statement will compile.
  The compiler fails if the program does not compile. }

program t6p8p3p5d1(output);
var
   i:integer;
begin
   i:=1;
   case i of
   1:
   end;
   writeln(' PASS...6.8.3.5-1, CASE');
end.
```

```
{TEST 6.8.3.5-2, CLASS=QUALITY}

{ This test checks that the case constants are of the same type
  as the case index. A compiler of good quality will detect that
  one path of the case statement cannot be taken and issue a
  warning message.
  The case-index in this test is a subrange and the case-constants
  are of the base type of the subrange. }

program t6p8p3p5d2(output);
type
   day=(mon,tue,wed);
var
   a:integer;
   d:mon..tue;
begin
   for d:=mon to tue do
      case d of
      mon: a:=1;
      tue: a:=2;
      wed: a:=3;   { could give a warning }
      end;
   writeln(' QUALITY TEST - WARNINGS FOR IMPOSSIBLE CASES');
   writeln(' PASS...6.8.3.5-2, CASE CONSTANTS');
end.


{TEST 6.8.3.5-3, CLASS=DEVIANCE}

{ This test checks that the constants of a case statement
  cannot be strings. The compiler deviates if the program
  compiles and prints DEVIATES. }

program t6p8p3p5d3(output);
var
   a:char;
   i:integer;
begin
   for a:= 'a' to 'd' do
      case a of
      'a': i:=1;
      'b': i:=i+1;
      'c': i:=i+1;
      'de': i:=i+1;
      end;
   writeln(' DEVIATES...6.8.3.5-3, CASE');
end.
```

```
{TEST 6.8.3.5-4, CLASS=CONFORMANCE}                                    00882500
                                                                       00882600
{ This test checks that a compiler handles a sparse case adequately    00882700
  Most compilers issue a jump table for a case, regardless             00882800
  of its structure. It is easy to optimise case statements             00882900
  to generate conditional statements if this is more compact.          00883000
  The compiler fails if the program does not compile                   00883100
  or the program fails in execution. }                                 00883200
                                                                       00883300
program t6p8p3p5d4(output);                                            00883400
var                                                                    00883500
   i,j:integer;                                                        00883600
begin                                                                  00883700
   i:=-1000;                                                           00883800
   for j:=1 to 2 do                                                    00883900
      case i of                                                        00884000
      -1000: i:=-i;                                                    00884100
      1000: writeln(' PASS...6.8.3.5-4, SPARSE CASE');                 00884200
      end;                                                             00884300
end.                                                                   00884400


{TEST 6.8.3.5-5, CLASS=ERRORHANDLING}                                  00884500
                                                                       00884600
{ This test checks the type of error produced when the case            00884700
  statement does not contain a constant of the selected value.         00884800
  An execution error should be produced. }                             00884900
                                                                       00885000
program t6p8p3p5d5(output);                                            00885100
var                                                                    00885200
   i:integer;                                                          00885300
begin                                                                  00885400
   i:=0;                                                               00885500
   case i of                                                           00885600
   -3,3: writeln(' FAIL...6.8.3.5-5, CASE');                           00885700
   end;                                                                00885800
   writeln(' ERROR NOT DETECTED...6.8.3.5-5, CASE CONSTANT');          00885900
end.                                                                   00886000


{TEST 6.8.3.5-6, CLASS=ERRORHANDLING}                                  00886100
                                                                       00886200
{ This test is similar to the previous one - a case statement          00886300
  is given without a case-constant of the selected value. This         00886400
  time the value is a long way outside the case. An error should       00886500
  be produced at execution time. }                                     00886600
                                                                       00886700
program t6p8p3p5d6(output);                                            00886800
var                                                                    00886900
   i:integer;                                                          00887000
begin                                                                  00887100
   i:=1000;                                                            00887200
   case i of                                                           00887300
   -3,3: writeln(' FAIL...6.8.3.5-6, CASE');                           00887400
   end;                                                                00887500
   writeln(' ERROR NOT DETECTED...6.8.3.5-6, CASE CONSTANT');          00887600
end.                                                                   00887700
```

```
{TEST 6.8.3.5-7,CLASS=ERRORHANDLING}                                   00887800
                                                                       00887900
{ This test contains an invalid real case-constant with an integer     00888000
  case expression. If the program compiles the effect at run-time      00888100
  could be curious. }                                                  00888200
                                                                       00888300
program t6p8p3p5d7(output);                                            00888400
var                                                                    00888500
   a,i:integer;                                                        00888600
begin                                                                  00888700
   for i:=1 to 4 do                                                    00888800
      case i of                                                        00888900
      1,2: a:=1;                                                       00889000
      2.5: writeln(' DEVIATES...6.8.3.5-7. CASE');                     00889100
      3:   a:=2;                                                       00889200
      4e0: writeln(' DEVIATES...6.8.3.5-7, CASE');                     00889300
      end;                                                             00889400
   writeln(' DEVIATES...6.8.3.5-7, CASE');                             00889500
end.                                                                   00889600
```

```
{TEST 6.8.3.5-8, CLASS=QUALITY}

{ This test checks a large populated case statement to check the
  limit on the size of code is not a serious one.
  The compiler has a small limit on the size of the case
  statement if the program does not compile and print PASS. }

program t6p8p3p5d8(output);
var
   sum:integer;
   i:0..255;
begin
   sum :=0;
   for i:=0 to 255 do
      case i of
       0 : sum := sum + i;
       1 : sum := sum + i;
       2 : sum := sum + i;
       3 : sum := sum + i;
       4 : sum := sum + i;
       5 : sum := sum + i;
       6 : sum := sum + i;
       7 : sum := sum + i;
       8 : sum := sum + i;
       9 : sum := sum + i;
      10 : sum := sum + i;
      11 : sum := sum + i;
      12 : sum := sum + i;
      13 : sum := sum + i;
      14 : sum := sum + i;
      15 : sum := sum + i;
      16 : sum := sum + i;
      17 : sum := sum + i;
      18 : sum := sum + i;
      19 : sum := sum + i;
      20 : sum := sum + i;
      21 : sum := sum + i;
      22 : sum := sum + i;
      23 : sum := sum + i;
      24 : sum := sum + i;
      25 : sum := sum + i;
      26 : sum := sum + i;
      27 : sum := sum + i;
      28 : sum := sum + i;
      29 : sum := sum + i;
      30 : sum := sum + i;
      31 : sum := sum + i;
      32 : sum := sum + i;
      33 : sum := sum + i;
      34 : sum := sum + i;
      35 : sum := sum + i;
      36 : sum := sum + i;
      37 : sum := sum + i;
      38 : sum := sum + i;
      39 : sum := sum + i;
      40 : sum := sum + i;
      41 : sum := sum + i;
      42 : sum := sum + i;
      43 : sum := sum + i;
      44 : sum := sum + i;
      45 : sum := sum + i;
      46 : sum := sum + i;
      47 : sum := sum + i;
      48 : sum := sum + i;
      49 : sum := sum + i;
      50 : sum := sum + i;
      51 : sum := sum + i;
      52 : sum := sum + i;
      53 : sum := sum + i;
      54 : sum := sum + i;
      55 : sum := sum + i;
      56 : sum := sum + i;
      57 : sum := sum + i;
      58 : sum := sum + i;
      59 : sum := sum + i;
      60 : sum := sum + i;
      61 : sum := sum + i;
      62 : sum := sum + i;
      63 : sum := sum + i;
      64 : sum := sum + i;
      65 : sum := sum + i;
      66 : sum := sum + i;
      67 : sum := sum + i;
      68 : sum := sum + i;
      69 : sum := sum + i;
      70 : sum := sum + i;
      71 : sum := sum + i;
      72 : sum := sum + i;
      73 : sum := sum + i;
      74 : sum := sum + i;
      75 : sum := sum + i;
      76 : sum := sum + i;
      77 : sum := sum + i;
      78 : sum := sum + i;
      79 : sum := sum + i;
      80 : sum := sum + i;
      81 : sum := sum + i;
      82 : sum := sum + i;
      83 : sum := sum + i;
      84 : sum := sum + i;
      85 : sum := sum + i;
      86 : sum := sum + i;
      87 : sum := sum + i;
      88 : sum := sum + i;
      89 : sum := sum + i;
      90 : sum := sum + i;
      91 : sum := sum + i;
      92 : sum := sum + i;
      93 : sum := sum + i;
      94 : sum := sum + i;
      95 : sum := sum + i;
      96 : sum := sum + i;
      97 : sum := sum + i;
      98 : sum := sum + i;
      99 : sum := sum + i;
     100 : sum := sum + i;
     101 : sum := sum + i;
     102 : sum := sum + i;
     103 : sum := sum + i;
     104 : sum := sum + i;
     105 : sum := sum + i;
     106 : sum := sum + i;
```

```
107 : sum := sum + i;                                                00901900
108 : sum := sum + i;                                                00902000
109 : sum := sum + i;                                                00902100
110 : sum := sum + i;                                                00902200
111 : sum := sum + i;                                                00902300
112 : sum := sum + i;                                                00902400
113 : sum := sum + i;                                                00902500
114 : sum := sum + i;                                                00902600
115 : sum := sum + i;                                                00902700
116 : sum := sum + i;                                                00902800
117 : sum := sum + i;                                                00902900
118 : sum := sum + i;                                                00903000
119 : sum := sum + i;                                                00903100
120 : sum := sum + i;                                                00903200
121 : sum := sum + i;                                                00903300
122 : sum := sum + i;                                                00903400
123 : sum := sum + i;                                                00903500
124 : sum := sum + i;                                                00903600
125 : sum := sum + i;                                                00903700
126 : sum := sum + i;                                                00903800
127 : sum := sum + i;                                                00903900
128 : sum := sum + i;                                                00904000
129 : sum := sum + i;                                                00904100
130 : sum := sum + i;                                                00904200
131 : sum := sum + i;                                                00904300
132 : sum := sum + i;                                                00904400
133 : sum := sum + i;                                                00904500
134 : sum := sum + i;                                                00904600
135 : sum := sum + i;                                                00904700
136 : sum := sum + i;                                                00904800
137 : sum := sum + i;                                                00904900
138 : sum := sum + i;                                                00905000
139 : sum := sum + i;                                                00905100
140 : sum := sum + i;                                                00905200
141 : sum := sum + i;                                                00905300
142 : sum := sum + i;                                                00905400
143 : sum := sum + i;                                                00905500
144 : sum := sum + i;                                                00905600
145 : sum := sum + i;                                                00905700
146 : sum := sum + i;                                                00905800
147 : sum := sum + i;                                                00905900
148 : sum := sum + i;                                                00906000
149 : sum := sum + i;                                                00906100
150 : sum := sum + i;                                                00906200
151 : sum := sum + i;                                                00906300
152 : sum := sum + i;                                                00906400
153 : sum := sum + i;                                                00906500
154 : sum := sum + i;                                                00906600
155 : sum := sum + i;                                                00906700
156 : sum := sum + i;                                                00906800
157 : sum := sum + i;                                                00906900
158 : sum := sum + i;                                                00907000
159 : sum := sum + i;                                                00907100
160 : sum := sum + i;                                                00907200
161 : sum := sum + i;                                                00907300
162 : sum := sum + i;                                                00907400
163 : sum := sum + i;                                                00907500
164 : sum := sum + i;                                                00907600
165 : sum := sum + i;                                                00907700
166 : sum := sum + i;                                                00907800
167 : sum := sum + i;                                                00907900
168 : sum := sum + i;                                                00908000
169 : sum := sum + i;                                                00908100
170 : sum := sum + i;                                                00908200
171 : sum := sum + i;                                                00908300
172 : sum := sum + i;                                                00908400
173 : sum := sum + i;                                                00908500
174 : sum := sum + i;                                                00908600
175 : sum := sum + i;                                                00908700
176 : sum := sum + i;                                                00908800
177 : sum := sum + i;                                                00908900
178 : sum := sum + i;                                                00909000
179 : sum := sum + i;                                                00909100
180 : sum := sum + i;                                                00909200
181 : sum := sum + i;                                                00909300
182 : sum := sum + i;                                                00909400
183 : sum := sum + i;                                                00909500
184 : sum := sum + i;                                                00909600
185 : sum := sum + i;                                                00909700
186 : sum := sum + i;                                                00909800
187 : sum := sum + i;                                                00909900
188 : sum := sum + i;                                                00910000
189 : sum := sum + i;                                                00910100
190 : sum := sum + i;                                                00910200
191 : sum := sum + i;                                                00910300
192 : sum := sum + i;                                                00910400
193 : sum := sum + i;                                                00910500
194 : sum := sum + i;                                                00910600
195 : sum := sum + i;                                                00910700
196 : sum := sum + i;                                                00910800
197 : sum := sum + i;                                                00910900
198 : sum := sum + i;                                                00911000
199 : sum := sum + i;                                                00911100
200 : sum := sum + i;                                                00911200
201 : sum := sum + i;                                                00911300
202 : sum := sum + i;                                                00911400
203 : sum := sum + i;                                                00911500
204 : sum := sum + i;                                                00911600
205 : sum := sum + i;                                                00911700
206 : sum := sum + i;                                                00911800
207 : sum := sum + i;                                                00911900
208 : sum := sum + i;                                                00912000
209 : sum := sum + i;                                                00912100
210 : sum := sum + i;                                                00912200
211 : sum := sum + i;                                                00912300
212 : sum := sum + i;                                                00912400
213 : sum := sum + i;                                                00912500
214 : sum := sum + i;                                                00912600
215 : sum := sum + i;                                                00912700
216 : sum := sum + i;                                                00912800
217 : sum := sum + i;                                                00912900
218 : sum := sum + i;                                                00913000
219 : sum := sum + i;                                                00913100
220 : sum := sum + i;                                                00913200
221 : sum := sum + i;                                                00913300
222 : sum := sum + i;                                                00913400
223 : sum := sum + i;                                                00913500
224 : sum := sum + i;                                                00913600
225 : sum := sum + i;                                                00913700
226 : sum := sum + i;                                                00913800
227 : sum := sum + i;                                                00913900
228 : sum := sum + i;                                                00914000
```

```
      229 : sum := sum + i;                                          00914100
      230 : sum := sum + i;                                          00914200
      231 : sum := sum + i;                                          00914300
      232 : sum := sum + i;                                          00914400
      233 : sum := sum + i;                                          00914500
      234 : sum := sum + i;                                          00914600
      235 : sum := sum + i;                                          00914700
      236 : sum := sum + i;                                          00914800
      237 : sum := sum + i;                                          00914900
      238 : sum := sum + i;                                          00915000
      239 : sum := sum + i;                                          00915100
      240 : sum := sum + i;                                          00915200
      241 : sum := sum + i;                                          00915300
      242 : sum := sum + i;                                          00915400
      243 : sum := sum + i;                                          00915500
      244 : sum := sum + i;                                          00915600
      245 : sum := sum + i;                                          00915700
      246 : sum := sum + i;                                          00915800
      247 : sum := sum + i;                                          00915900
      248 : sum := sum + i;                                          00916000
      249 : sum := sum + i;                                          00916100
      250 : sum := sum + i;                                          00916200
      251 : sum := sum + i;                                          00916300
      252 : sum := sum + i;                                          00916400
      253 : sum := sum + i;                                          00916500
      254 : sum := sum + i;                                          00916600
      255 : sum := sum + i;                                          00916700
      end;                                                           00916800
   writeln(' QUALITY TEST - SIZE OF CASE STATEMENT');                 00916900
   if sum = 32640 then                                               00917000
      writeln(' PASS...6.8.3.5-8')                                   00917100
   else                                                              00917200
      writeln(' FAIL...6.8.3.5-8');                                  00917300
end.                                                                 00917400


{TEST 6.8.3.5-9, CLASS=DEVIANCE}                                     00917500

{ This test checks that the compiler detects that case-constants     00917600
  and the case-index are of different types.                         00917700
  The compiler deviates if the program compiles and the program      00917800
  prints deviates. }                                                 00917900

program t6p8p3p5d9(output);                                          00918000
var                                                                  00918100
   i,counter:integer;                                                00918200
begin                                                                00918300
   counter:= 0;                                                      00918400
   for i:= 1 to 4 do                                                 00918500
      case i of                                                      00918600
      1: counter:=counter+1;                                         00918700
      2.0: counter:=counter+1;                                       00918800
      3: counter:=counter+1;                                         00918900
      4e0: counter:=counter+1;                                       00919000
      end;                                                           00919100
   if counter=4 then                                                 00919200
      writeln(' DEVIATES...6.8.3.5-9, CASE CONSTANTS')               00919300
   else                                                              00919400
      writeln(' FAILS...6.8.3.5-9, CASE CONSTANTS');                 00919500
end.                                                                 00919600
                                                                     00919700
                                                                     00919800
```

```
{TEST 6.8.3.5-10, CLASS=DEVIANCE}                                    00919900
                                                                     00920000
{ This test checks that the compiler detects real case constants     00920100
  and a real case index, even when the values are integers.          00920200
  The compiler fails if the program compiles and the program         00920300
  prints FAILS. }                                                    00920400
                                                                     00920500
program t6p8p3p5d10(output);                                         00920600
var                                                                  00920700
   i,counter:integer;                                                00920800
   r:real;                                                           00920900
begin                                                                00921000
   counter:= 0;                                                      00921100
   for i:= 1 to 4 do                                                 00921200
   begin                                                             00921300
      r:=i;                                                          00921400
      case r of                                                      00921500
      1.0: counter:=counter+1;                                       00921600
      2.0: counter:=counter+1;                                       00921700
      3.0: counter:=counter+1;                                       00921800
      4e0: counter:=counter+1;                                       00921900
      end;                                                           00922000
   end;                                                              00922100
   if counter=4 then                                                 00922200
      writeln(' DEVIANCE...6.8.3.5-10, CASE CONSTANTS')              00922300
   else                                                              00922400
      writeln(' FAILS...6.8.3.5-10, CASE CONSTANTS');                00922500
end.                                                                 00922600


{TEST 6.8.3.5-11, CLASS=DEVIANCE}                                    00922700
                                                                     00922800
{ This test checks that the compiler detects that a case index       00922900
  and the case constants are of different types.                     00923000
  The compiler deviates if the program compiles and the program      00923100
  prints DEVIATES. }                                                 00923200
                                                                     00923300
program t6p8p3p5d11(output);                                         00923400
var                                                                  00923500
   i,counter:integer;                                                00923600
   r:real;                                                           00923700
begin                                                                00923800
   counter:= 0;                                                      00923900
   for i:= 1 to 4 do                                                 00924000
   begin                                                             00924100
      r:=i;                                                          00924200
      case r of                                                      00924300
      1: counter:=counter+1;                                         00924400
      2: counter:=counter+1;                                         00924500
      3: counter:=counter+1;                                         00924600
      4: counter:=counter+1;                                         00924700
      end;                                                           00924800
   end;                                                              00924900
   if counter=4 then                                                 00925000
      writeln(' DEVIATES...6.8.3.5-11, CASE CONSTANTS')              00925100
   else                                                              00925200
      writeln(' PASS...6.8.3.5-11, CASE CONSTANTS');                 00925300
end.                                                                 00925400
```

```
{TEST 6.8.3.5-12, CLASS=DEVIANCE}                                    00925500
                                                                     00925600
{ Some processors allow subrange-like lists to be used as case-constant 00925700
  elements. This test checks to see if this is allowed. It is        00925800
  not standard Pascal. The compiler deviates if the program          00925900
  compiles and prints DEVIATES. }                                    00926000
                                                                     00926100
program t6p8p3p5d12(output);                                         00926200
var                                                                  00926300
   thing:(a,b,c,d,e,f);                                              00926400
begin                                                                00926500
   thing:=a;                                                         00926600
   while thing<>f do begin                                           00926700
      case thing of                                                  00926800
      a..d: thing := succ(thing);                                    00926900
      e:    thing:=f                                                 00927000
      end                                                            00927100
   end;                                                              00927200
   writeln(' DEVIATES...6.8.3.5-12, CASE CONSTANTS')                 00927300
end.                                                                 00927400


{TEST 6.8.3.5-13, CLASS=DEVIANCE}                                    00927500
                                                                     00927600
{ Similar to test 6.8.3.5-12, this test checks the subrange          00927700
  case extension, which may not be safely implemented. This program  00927800
  is utter confused garbage and indicates the kinds of checks        00927900
  needed. The compiler deviates if the program compiles and          00928000
  prints DEVIATES. }                                                 00928100
                                                                     00928200
program t6p8p3p5d13(output);                                         00928300
var                                                                  00928400
   t,thing:(a,b,c,d,e,f,g,h);                                        00928500
begin                                                                00928600
   for thing:=a to g do begin                                        00928700
      case thing of                                                  00928800
      a..e: t:=thing;                                                00928900
      d..g: t:=succ(thing);                                          00929000
      b:    t:=pred(thing)                                           00929100
      end                                                            00929200
   end;                                                              00929300
   writeln(' DEVIATES...6.8.3.5-13, CASE CONSTANTS')                 00929400
end.                                                                 00929500
```

```
{TEST 6.8.3.5-14, CLASS=EXTENSION, SUBCLASS=CONFORMANCE}             00929600
                                                                     00929700
{ This test checks whether an otherwise clause in a case statement   00929800
  is accepted. The convention is that adopted at the UCSD Pascal     00929900
  workshop in July 1978. The extension is accepted if the program    00930000
  compiles and prints EXTENSION - PASS }                             00930100
                                                                     00930200
program t6p8p3p5d14(output);                                         00930300
var                                                                  00930400
   i,j,k,counter:integer;                                            00930500
begin                                                                00930600
   counter:=0;                                                       00930700
   for i:=0 to 10 do                                                 00930800
      case i of                                                      00930900
      1,3,5,7,9:                                                     00931000
         counter:=counter+1;                                         00931100
      otherwise                                                      00931200
         j:=counter;                                                 00931300
         k:=j;                                                       00931400
      end;                                                           00931500
   if (counter = 5) then                                             00931600
      writeln(' EXTENSION - PASS...6.8.3.5-14, OTHERWISE')           00931700
   else                                                              00931800
      writeln(' EXTENSION - FAIL...6.8.3.5-14, OTHERWISE');          00931900
end.                                                                 00932000


{TEST 6.8.3.7-1, CLASS=CONFORMANCE}                                  00932100
                                                                     00932200
{ This test checks that a repeat loop is executed at least once.     00932300
  The compiler fails if the program prints FAILS. }                  00932400
                                                                     00932500
program t6p8p3p7d1(output);                                          00932600
var                                                                  00932700
   counter:integer;                                                  00932800
   bool:boolean;                                                     00932900
begin                                                                00933000
   bool:=true;                                                       00933100
   counter:=0;                                                       00933200
   repeat                                                            00933300
      counter:=counter+1                                             00933400
   until bool;                                                       00933500
   if(counter=1) then                                                00933600
      writeln(' PASS...6.8.3.7-1, REPEAT')                           00933700
   else                                                              00933800
      writeln(' FAIL...6.8.3.7-1, REPEAT');                          00933900
end.                                                                 00934000
```

```
{TEST 6.8.3.7-2, CLASS=CONFORMANCE}                                    00934100
                                                                       00934200
{ This test checks that a loop containing no statements               00934300
  is executed until the expression is true.                            00934400
  The compiler fails if the program does not compile or the            00934500
  program prints FAIL. }                                               00934600
                                                                       00934700
program t6p8p3p7d2(output);                                            00934800
var                                                                    00934900
  a:integer;                                                           00935000
                                                                       00935100
function bool : boolean;                                               00935200
begin                                                                  00935300
   a:=a+1;                                                             00935400
   bool := a>=5;                                                       00935500
end;                                                                   00935600
                                                                       00935700
begin                                                                  00935800
   a:=0;                                                               00935900
   repeat                                                              00936000
   until bool;                                                         00936100
   if (a=5) then                                                       00936200
      writeln(' PASS...6.8.3.7-2, EMPTY REPEAT')                       00936300
   else                                                                00936400
      writeln(' FAIL...6.8.3.7-2, EMPTY REPEAT');                      00936500
end.                                                                   00936600


{TEST 6.8.3.7-3, CLASS=CONFORMANCE}                                    00936700
                                                                       00936800
{ This test clecks that an apparently infinite loop is allowed         00936900
  by the compiler. Some compilers may detect the loop as being         00937000
  infinite. The compiler fails if the program does not compile. }      00937100
                                                                       00937200
program t6p8p3p7d3(output);                                            00937300
label                                                                  00937400
   100;                                                                00937500
const                                                                  00937600
   eternity = false;                                                   00937700
var                                                                    00937800
   i:integer;                                                          00937900
begin                                                                  00938000
   i:=0;                                                               00938100
   repeat                                                              00938200
      i:=i+1;                                                          00938300
      if (i>50) then                                                   00938400
         goto 100;                                                     00938500
   until eternity;                                                     00938600
100:                                                                   00938700
   writeln(' PASS...6.8.3.7-3, REPEAT');                               00938800
end.                                                                   00938900
```

```
{TEST 6.8.3.8-1, CLASS=CONFORMANCE}                                    00939000
                                                                       00939100
{ This test checks that a while loop is not entered                    00939200
  if the initial value of the boolean expression is false.             00939300
  The compiler fails if the program prints FAIL. }                     00939400
                                                                       00939500
program t6p8p3p8d1(output);                                            00939600
var                                                                    00939700
   bool:boolean;                                                       00939800
   counter:integer;                                                    00939900
begin                                                                  00940000
   counter:=0;                                                         00940100
   bool:=false;                                                        00940200
   while bool do                                                       00940300
   begin                                                               00940400
      counter:=counter+1;                                              00940500
      bool:=false;                                                     00940600
   end;                                                                00940700
   if (counter=0) then                                                 00940800
      writeln(' PASS...6.8.3.8-1, WHILE')                              00940900
   else                                                                00941000
      writeln(' FAIL...6.8.3.8-1, WHILE');                             00941100
end.                                                                   00941200


{TEST 6.8.3.8-2, CLASS=CONFORMANCE}                                    00941300
                                                                       00941400
{ This test checks that the compiler will accept a while               00941500
  loop containing no statements. The compiler fails if the             00941600
  program does not compile or the program prints FAIL. }               00941700
                                                                       00941800
program t6p8p3p8d2(output);                                            00941900
var                                                                    00942000
   a:integer;                                                          00942100
                                                                       00942200
function bool:boolean;                                                 00942300
begin                                                                  00942400
   a:=a+1;                                                             00942500
   bool:= (a>=5);                                                      00942600
end;                                                                   00942700
                                                                       00942800
begin                                                                  00942900
   a:=0;                                                               00943000
   while not bool do ;                                                 00943100
   if (a=5) then                                                       00943200
      writeln(' PASS...6.8.3.8-2, EMPTY WHILE')                        00943300
   else                                                                00943400
      writeln(' FAIL...6.8.3.8-2, EMPTY WHILE');                       00943500
end.                                                                   00943600
```

```
{TEST 6.8.3.9-1, CLASS=CONFORMANCE}                                    00943700
                                                                       00943800
{ This program checks that assignment follows the  evaluation of       00943900
  both expressions  in a for statement.                                00944000
  The compiler fails if the program prints FAIL. }                     00944100
                                                                       00944200
program t6p8p3p9d1(output);                                            00944300
var                                                                    00944400
   i,j:integer;                                                        00944500
begin                                                                  00944600
   i:=1;                                                               00944700
   j:=0;                                                               00944800
   for i:= (i+1) to (i+10) do                                          00944900
   begin                                                               00945000
      j:=j+1;                                                          00945100
      writeln(i);                                                      00945200
   end;                                                                00945300
   if (j=10) then                                                      00945400
      writeln(' PASS...6.8.3.9-1, FOR')                                00945500
   else                                                                00945600
      writeln(' FAIL...6.8.3.9-1, FOR');                               00945700
end.                                                                   00945800


{TEST 6.8.3.9-2, CLASS=DEVIANCE}                                       00945900
                                                                       00946000
{ This program checks that an assignment cannot be made to a           00946100
  for statement control variable.                                      00946200
  The compiler deviates if the program compiles and prints             00946300
  DEVIATES. }                                                          00946400
                                                                       00946500
program t6p8p3p9d2(output);                                            00946600
var                                                                    00946700
  i,j:integer;                                                         00946800
begin                                                                  00946900
   j:=0;                                                               00947000
   for i:=1 to 10 do                                                   00947100
   begin                                                               00947200
      j:=j+1;                                                          00947300
      i:=i+1;                                                          00947400
      writeln(j,i);                                                    00947500
   end;                                                                00947600
   writeln(' DEVIATES...6.8.3.9-2, FOR');                              00947700
end.                                                                   00947800
```

```
{TEST 6.8.3.9-3, CLASS=DEVIANCE}                                       00947900
                                                                       00948000
{ This test checks that an error is produced when an                   00948100
  assignment is made to a for statement control variable.              00948200
  The compiler deviates if the program compiles and prints             00948300
  DEVIATES. }                                                          00948400
                                                                       00948500
program t6p8p3p9d3(output);                                            00948600
var                                                                    00948700
   i,j:integer;                                                        00948800
                                                                       00948900
procedure nasty (var n:integer);                                       00949000
begin                                                                  00949100
   n:=n+1;                                                             00949200
end;                                                                   00949300
                                                                       00949400
begin                                                                  00949500
   j:=0;                                                               00949600
   for i:=1 to 10 do                                                   00949700
   begin                                                               00949800
      j:=j+1;                                                          00949900
      nasty(i);                                                        00950000
   end;                                                                00950100
   writeln(' DEVIATES...6.8.3.9-3, FOR');                              00950200
end.                                                                   00950300


{TEST 6.8.3.9-4, CLASS=DEVIANCE}                                       00950400
                                                                       00950500
{ This program tests that an error is produced when an assignment      00950600
  is made to a for statement control variable.                         00950700
  The compiler deviates if the program compiles and prints             00950800
  DEVIATES. }                                                          00950900
                                                                       00951000
program t6p8p3p9d4(output);                                            00951100
var                                                                    00951200
   i,j:integer;                                                        00951300
                                                                       00951400
procedure verynasty;                                                   00951500
begin                                                                  00951600
   i:=i+1;                                                             00951700
end;                                                                   00951800
                                                                       00951900
begin                                                                  00952000
   j:=0;                                                               00952100
   for i:= 1 to 10 do                                                  00952200
   begin                                                               00952300
      j:=j+1;                                                          00952400
      verynasty;                                                       00952500
   end;                                                                00952600
   writeln(' DEVIATES...6.8.3.9-4, FOR');                              00952700
end.                                                                   00952800
```

```
{TEST 6.8.3.9-5, CLASS=ERRORHANDLING}                                   00952900
                                                                        00953000
{ This test checks that the use of a for statement control              00953100
  variable after the completion of the for statement, and               00953200
  without an intervening assignment is detected. }                      00953300
                                                                        00953400
program t6p8p3p9d5(output);                                             00953500
var                                                                     00953600
   i,j,k,m:integer;                                                     00953700
begin                                                                   00953800
   i:=100;                                                              00953900
   j:=1;                                                                00954000
   k:=10;                                                               00954100
   m:=0;                                                                00954200
   for i:=j to k do                                                     00954300
   begin                                                                00954400
      m:=m+1;                                                           00954500
   end;                                                                 00954600
   writeln('  THE VALUE OF I =',i);                                     00954700
   writeln(' ERROR NOT DETECTED...6.8.3.9-5, FOR');                     00954800
end.                                                                    00954900
```

```
{TEST 6.8.3.9-6, CLASS=ERRORHANDLING}                                   00955000
                                                                        00955100
{ This program uses a for statement control variable after a            00955200
  for loop which is not entered. The control variable should be         00955300
  undefined after the for statement. }                                  00955400
                                                                        00955500
program t6p8p3p9d6(output);                                             00955600
var                                                                     00955700
   i,j,k,m:integer;                                                     00955800
begin                                                                   00955900
   i:=100;                                                              00956000
   k:=1;                                                                00956100
   m:=0;                                                                00956200
   j:=10;                                                               00956300
   for i:=j to k do                                                     00956400
   begin                                                                00956500
      m:=m+1;                                                           00956600
   end;                                                                 00956700
   writeln(' THE VALUE OF I =',i);                                      00956800
   writeln(' ERROR NOT DETECTED...6.8.3.9-6, FOR');                     00956900
end.                                                                    00957000
```

```
{TEST 6.8.3.9-7, CLASS=CONFORMANCE}                                     00957100
                                                                        00957200
{ This test checks that extreme values may be used in a for             00957300
  loop. This will break a simply implemented for loop.                  00957400
  In some compilers the succ test may fail at the last increment        00957500
  and cause wraparound(overflow) - leading to an infinite loop. }       00957600
                                                                        00957700
program t6p8p3p9d7(output);                                             00957800
var                                                                     00957900
   i,j:integer;                                                         00958000
begin                                                                   00958100
   j:=0;                                                                00958200
   for i:= (maxint-10) to maxint do                                     00958300
     j:=j+1;                                                            00958400
   for i:= (-maxint+10) downto -maxint do                               00958500
      j:=j+1;                                                           00958600
   if j = 22 then                                                       00958700
      writeln(' PASS...6.8.3.9-7, FOR LOOP')                            00958800
   else                                                                 00958900
      writeln(' FAIL...6.8.3.9-7, FOR LOOP');                           00959000
end.                                                                    00959100
```

```
{TEST 6.8.3.9-8, CLASS=CONFORMANCE}                                     00959200
                                                                        00959300
{ This program checks that a control variable of a for statement        00959400
  is not undefined if the for statement is left via a goto              00959500
  statement. The compiler fails if the program does not compile         00959600
  or the program prints FAIL. }                                         00959700
                                                                        00959800
program t6p8p3p9d8(output);                                             00959900
label 100;                                                              00960000
var                                                                     00960100
   i,j:integer;                                                         00960200
begin                                                                   00960300
   j:=1;                                                                00960400
   for i:=1 to 10 do                                                    00960500
   begin                                                                00960600
      if (j=5) then                                                     00960700
         goto 100;                                                      00960800
      j:=j+1;                                                           00960900
   end;                                                                 00961000
100:                                                                    00961100
   if i=j then                                                          00961200
      writeln(' PASS...6.8.3.9-8, FOR')                                 00961300
   else                                                                 00961400
      writeln(' FAIL...6.8.3.9-8, FOR');                                00961500
end.                                                                    00961600
```

```
{TEST 6.8.3.9-9, CLASS=DEVIANCE}                                        00961700
                                                                        00961800
{ This program tests whether a non local variable at an intermediate    00961900
  level can be used as a for statement control variable.                00962000
  The program deviates if the program compiles and prints               00962100
  DEVIATES. }                                                           00962200
                                                                        00962300
program t6p8p3p9d9(output);                                             00962400
                                                                        00962500
procedure p;                                                            00962600
var                                                                     00962700
   i:integer;                                                           00962800
                                                                        00962900
   procedure loop;                                                      00963000
   var                                                                  00963100
      j:integer;                                                        00963200
   begin                                                                00963300
      j:=0;                                                             00963400
      for i:=1 to 10 do                                                 00963500
         j:=j+1;                                                        00963600
   end;                                                                 00963700
begin                                                                   00963800
   loop                                                                 00963900
end;                                                                    00964000
                                                                        00964100
begin                                                                   00964200
   p;                                                                   00964300
   writeln(' DEVIATES...6.8.3.9-9, FOR');                               00964400
end.                                                                    00964500


{TEST 6.8.3.9-10, CLASS=DEVIANCE}                                       00964600
                                                                        00964700
{ This program tests whether a real number can be assigned to           00964800
  a for statement control variable. The compiler deviates               00964900
  if the program compiles and prints DEVIATES. }                        00965000
                                                                        00965100
program t6p8p3p9d10(output);                                            00965200
var                                                                     00965300
   i:integer;                                                           00965400
   counter:integer;                                                     00965500
begin                                                                   00965600
   counter:=0;                                                          00965700
   for i:=0.0 to 3.5 do                                                 00965800
      counter:=counter+1;                                               00965900
   if(counter=4) then                                                   00966000
      writeln(' DEVIATES...6.8.3.9-10, FOR EXPRESSION ROUNDED')         00966100
   else                                                                 00966200
      writeln(' DEVIATES...6.8.3.9-10, FOR EXPRESSION TRUNCATED');      00966300
end.                                                                    00966400
```

```
{TEST 6.8.3.9-11, CLASS=DEVIANCE}                                       00966500
                                                                        00966600
{ This test checks whether a for statement control variable             00966700
  can be a component variable.                                          00966800
  The compiler deviates if the program compiles and prints              00966900
  DEVIATES. }                                                           00967000
                                                                        00967100
program t6p8p3p9d11(output);                                            00967200
var                                                                     00967300
   rec:record                                                           00967400
         i,j:integer;                                                   00967500
       end;                                                             00967600
begin                                                                   00967700
   for rec.i:=0 to 10 do                                                00967800
      rec.j := rec.i;                                                   00967900
   with rec do                                                          00968000
      for i := 0 to 10 do                                               00968100
         j:=i;                                                          00968200
   writeln(' DEVIATES...6.8.3.9-11, FOR');                              00968300
end.                                                                    00968400


{TEST 6.8.3.9-12, CLASS=DEVIATES}                                       00968500
                                                                        00968600
{ This test checks whether a for statement control variable             00968700
  can be a pointer variable.                                            00968800
  The compiler deviates if the program compiles and prints              00968900
  DEVIATES. }                                                           00969000
                                                                        00969100
program t6p8p3p9d12(output);                                            00969200
type                                                                    00969300
   int = ↑integer;                                                      00969400
var                                                                     00969500
   ptr:int;                                                             00969600
   j:integer;                                                           00969700
begin                                                                   00969800
   j:=0;                                                                00969900
   new(ptr);                                                            00970000
   for ptr↑ := 0 to 10 do                                               00970100
      j:=j+1;                                                           00970200
   writeln(' DEVIATES...6.8.3.9-12, FOR');                              00970300
end.                                                                    00970400
```

```
{TEST 6.8.3.9-13, CLASS=DEVIANCE}

{ This program tests whether a formal parameter  can be used
  as a for statement control variable.
  The program deviates if the program compiles and prints
  DEVIATES. }

program t6p8p3p9d13(output);

procedure p;
var
   i:integer;

   procedure loop(var i:integer);
   var
      j:integer;
   begin
      j:=0;
      for i:=1 to 10 do
         j:=j+1;
   end;
begin
   i:=10;
   loop(i)
end;

begin
   p;
   writeln(' DEVIATES...6.8.3.9-13, FOR');
end.
```

```
{TEST 6.8.3.9-14, CLASS=DEVIANCE}

{ This program tests whether a global variable (at program level)
  can be used as a for statement control variable.
  The program deviates if the program compiles and prints
  DEVIATES. }

program t6p8p3p9d14(output);
var
   i:integer;

procedure p;

   procedure loop;
   var
      j:integer;
   begin
      j:=0;
      for i:=1 to 10 do
         j:=j+1;
   end;
begin
   loop
end;

begin
   p;
   writeln(' DEVIATES...6.8.3.9-14, FOR');
end.
```

```
{TEST 6.8.3.9-15, CLASS=CONFORMANCE}

{ This program checks the order of evaluation of the limit expressions
  in a for statement.
  The compiler fails if the program prints FAIL. }

program t6p8p3p9d15(output);
var
   i,j,k:integer;

function f(var k:integer) : integer;
begin
   k:=k+1;
   f:=k;
end;

begin
   k:=0;
   j:=0;
   for i:=f(k) to f(k)+10 do
   begin
      j:=j+1;
      writeln(i);
   end;
   if (j=12) then
      writeln(' PASS...6.8.3.9-15, FOR')
   else
      writeln(' FAIL...6.8.3.9-15, FOR');
end.
```

```
{TEST 6.8.3.9-16, CLASS=DEVIANCE}                                    00979300
                                                                     00979400
{ This test checks the type of error produced when a for statement   00979500
  control variable value is read during the execution of the for     00979600
  statement. The compiler deviates if the program compiles and       00979700
  prints DEVIATES. }                                                 00979800
                                                                     00979900
program t6p8p3p9d16(output,f);                                       00980000
var                                                                  00980100
   f:text;                                                           00980200
   i,j:integer;                                                      00980300
begin                                                                00980400
   j:=0;                                                             00980500
   rewrite(f);                                                       00980600
   writeln(f,5,5,5,5,5);                                             00980700
   reset(f);                                                         00980800
   for i := 1 to 10 do                                               00980900
   begin                                                             00981000
      if (i<5) then                                                  00981100
         read(f,i);                                                  00981200
      j:=j+1;                                                        00981300
   end;                                                              00981400
   writeln(' DEVIATES...6.8.3.9-16, FOR');                           00981500
end.                                                                 00981600


{TEST 6.8.3.9-17, CLASS=ERRORHANDLING}                               00981700
                                                                     00981800
{ This test checks the type of error produced when two nested        00981900
  for statements use the same control variable. }                    00982000
                                                                     00982100
program t6p8p3p9d17(output);                                         00982200
var                                                                  00982300
   i,j:integer;                                                      00982400
begin                                                                00982500
   j:=0;                                                             00982600
   for i:=1 to 10 do                                                 00982700
      for i:=1 to 10 do                                              00982800
         j:=j+1;                                                     00982900
   writeln(' ERROR NOT DETECTED...6.8.3.9-17, FOR');                 00983000
end.                                                                 00983100
```

```
{TEST 6.8.3.9-18, CLASS=QUALITY}                                     00983200
                                                                     00983300
{ This test checks that the undefined state of a for-statement       00983400
  controlled variable when the loop is left has one or both of       00983500
  the following properties:                                          00983600
    (a) Range checks are not omitted on these variables in the       00983700
        supposition that its value is permissible, or                00983800
    (b) the value of the variable is in range of its type (in        00983900
        this specific implementation).                               00984000
  This test is not relevant if the use of the variable is            00984100
  prohibited. }                                                      00984200
                                                                     00984300
program t6p8p3p9d18(output);                                         00984400
type                                                                 00984500
   t=(red,green,blue,pink);                                          00984600
var                                                                  00984700
   i,j,k:t;                                                          00984800
   m:integer;                                                        00984900
begin                                                                00985000
   { i is a finite scalar. }                                         00985100
   i:=green;                                                         00985200
   j:=red;                                                           00985300
   k:=pink;                                                          00985400
   m:=0;                                                             00985500
   for i:=j to k do                                                  00985600
   begin                                                             00985700
      m:=m+1;                                                        00985800
   end;                                                              00985900
   writeln(' THE UNDEFINED ORDINAL VALUE OF I IS ',ord(i));          00986000
   writeln(' ERROR NOT DETECTED');                                   00986100
   write(' ITS SYMBOLIC VALUE IS ');                                 00986200
                                                                     00986300
   { A possible omission of the range check on the case statement    00986400
     may be disastrous if a wild jump occurs. }                      00986500
                                                                     00986600
   case i of                                                         00986700
   red: writeln('RED');                                              00986800
   green: writeln('GREEN');                                          00986900
   blue: writeln('BLUE');                                            00987000
   pink: writeln('PINK');                                            00987100
   end;                                                              00987200
   writeln(' JUST IN CASE THE RANGE ISNT CHECKED');                  00987300
end.                                                                 00987400
```

```
{TEST 6.8.3.9-19, CLASS=DEVIANCE}                                    00987500
                                                                     00987600
{ This test checks that compilers that permit the deviation          00987700
  (extension?) of allowing non-local control variables do so         00987800
  responsibly and do not introduce new insecurities.                 00987900
  This test checks that a nested for statement using the same control 00988000
  variable is detected. It is similar to test 6.8.3.9-14 but         00988100
  requires a degree of sophistication to detect this condition.      00988200
  The compiler deviates if the program prints DEVIATES.              00988300
  The program may loop endlessly under some compilers. }             00988400
                                                                     00988500
program t6p8p3p9d19(output);                                         00988600
var                                                                  00988700
   i:integer;                                                        00988800
                                                                     00988900
procedure p;                                                         00989000
   procedure q;                                                      00989100
      procedure r;                                                   00989200
         procedure s(var i:integer);                                 00989300
         begin                                                       00989400
            writeln(i);                                              00989500
         end;                                                        00989600
      begin                                                          00989700
         for i:= 5 downto 2 do                                       00989800
            s(i);                                                    00989900
      end;                                                           00990000
   begin                                                             00990100
      r                                                              00990200
   end;                                                              00990300
begin                                                                00990400
   q                                                                 00990500
end;                                                                 00990600
                                                                     00990700
begin                                                                00990800
   for i:= 1 to 6 do                                                 00990900
      p;                                                             00991000
   writeln(' DEVIATES...6.8.3.9-19, FOR')                            00991100
end.                                                                 00991200
```

```
{TEST 6.8.3.9-20, CLASS=QUALITY}                                     00991300
                                                                     00991400
{ This test checks that for statements may be nested to 15 levels    00991500
  The test may detect a small compiler limit, particularly           00991600
  those compilers that use a register for a control variable. }      00991700
                                                                     00991800
program t6p8p3p9d20(output);                                         00991900
var                                                                  00992000
   i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,i11,i12,i13,i14,i15:integer;       00992100
   j:integer;                                                        00992200
begin                                                                00992300
   for i1:=1 to 2 do                                                 00992400
     for i2:=1 to 2 do                                               00992500
        for i3:=1 to 2 do                                            00992600
           for i4:=1 to 2 do                                         00992700
              for i5:=1 to 2 do                                      00992800
                 for i6:=1 to 2 do                                   00992900
                    for i7:=1 to 2 do                                00993000
                       for i8:=1 to 2 do                             00993100
                          for i9:=1 to 2 do                          00993200
                             for i10:=1 to 2 do                      00993300
                                for i11:=1 to 2 do                   00993400
                                   for i12:=1 to 2 do                00993500
                                      for i13:=1 to 2 do             00993600
                                         for i14:=1 to 2 do          00993700
                                            for i15:=1 to 2 do       00993800
                                               j:=10;                00993900
   writeln(' FOR STATEMENT NESTED TO > 15 LEVELS...6.8.3.9-20')      00994000
end.                                                                 00994100
```

```
{TEST 6.8.3.10-1, CLASS=CONFORMANCE}                                    00994200
                                                                        00994300
{ This program checks the implementation of the with statement.         00994400
  The compiler fails if the program does not compile or it              00994500
  compiles and prints FAILS. }                                          00994600
                                                                        00994700
program t6p8p3p10d1(output);                                            00994800
var                                                                     00994900
   r1:record                                                            00995000
        a,b:integer                                                     00995100
      end;                                                              00995200
   r2:record                                                            00995300
        c,d:integer                                                     00995400
      end;                                                              00995500
   r3:record                                                            00995600
        e,f:integer                                                     00995700
      end;                                                              00995800
   counter:integer;                                                     00995900
begin                                                                   00996000
   counter:=0;                                                          00996100
   with r1 do                                                           00996200
      a:=5;                                                             00996300
   with r1,r2,r3 do                                                     00996400
   begin                                                                00996500
      e:=a;                                                             00996600
      c:=a                                                              00996700
   end;                                                                 00996800
   with r2 do                                                           00996900
      if c=5 then                                                       00997000
         counter:=counter+1;                                            00997100
   if r2.c=5 then                                                       00997200
      counter:=counter+1;                                               00997300
   if counter=2 then                                                    00997400
      writeln(' PASS 6.8.3.10-1, WITH')                                 00997500
   else                                                                 00997600
      writeln(' FAIL 6.8.3.10-1, WITH');                                00997700
end.                                                                    00997800
```

```
{TEST 6.8.3.10-2, CLASS=CONFORMANCE}                                    00997900
                                                                        00998000
{ This test checks that a field identifier is correctly                 00998100
  identified when a with statement is invoked.                          00998200
  The compiler fails if the program does not compile or the             00998300
  program prints FAILS. }                                               00998400
                                                                        00998500
program t6p8p3p10d2(output);                                            00998600
var                                                                     00998700
   r:record                                                             00998800
       i,j:integer                                                      00998900
     end;                                                               00999000
   i:integer;                                                           00999100
begin                                                                   00999200
   i:=10;                                                               00999300
   with r do                                                            00999400
      i:=5;                                                             00999500
   if (i=10) and (r.i=5) then                                           00999600
      writeln(' PASS 6.8.3.10-2, WITH')                                 00999700
   else                                                                 00999800
      writeln(' FAIL 6.8.3.10-2, WITH');                                00999900
end.                                                                    01000000
```

```
{TEST 6.8.3.10-3, CLASS=CONFORMANCE}                                    01000100
                                                                        01000200
{ This test checks that the record-variable-list is evaluated           01000300
  in the correct order. The compiler fails if the program does          01000400
  not compile or the program prints FAILS. }                            01000500
                                                                        01000600
program t6p8p3p10d3(output);                                            01000700
var                                                                     01000800
   r1:record                                                            01000900
        i,j,k:integer                                                   01001000
      end;                                                              01001100
   r2:record                                                            01001200
        i,j:integer                                                     01001300
      end;                                                              01001400
   r3:record                                                            01001500
        i:integer                                                       01001600
      end;                                                              01001700
begin                                                                   01001800
   with r1 do                                                           01001900
   begin                                                                01002000
      i:=0;                                                             01002100
      j:=0;                                                             01002200
      k:=0                                                              01002300
   end;                                                                 01002400
   with r2 do                                                           01002500
   begin                                                                01002600
      i:=0;                                                             01002700
      j:=0                                                              01002800
   end;                                                                 01002900
   with r3 do                                                           01003000
      i:=0;                                                             01003100
   with r1,r2,r3 do                                                     01003200
   begin                                                                01003300
      i:=5;                                                             01003400
      j:=6;                                                             01003500
      k:=7                                                              01003600
   end;                                                                 01003700
   if(r1.i=0) and (r1.j=0) and (r2.i=0) and (r1.k=7)                    01003800
      and (r2.j=6) and (r3.i=5) then                                    01003900
      writeln(' PASS 6.8.3.10-3, WITH EVALUATION')                      01004000
   else                                                                 01004100
      writeln(' FAIL 6.8.3.10-3, WITH EVALUATION');                     01004200
end.                                                                    01004300
```

```
{TEST 6.8.3.10-4, CLASS=CONFORMANCE}                                    01004400
                                                                        01004500
{ This test checks that the selection of a variable in the              01004600
  record-variable-list is performed before the component                01004700
  statement is executed. The compiler fails if the program              01004800
  does not compile or the program prints FAIL. }                        01004900
                                                                        01005000
program t6p8p3p10d4(output);                                            01005100
var                                                                     01005200
   a:array[1..2] of record                                              01005300
                      i,j:integer                                       01005400
                    end;                                                01005500
   k:integer;                                                           01005600
begin                                                                   01005700
   a[2].i:=5;                                                           01005800
   k:=1;                                                                01005900
   with a[k] do                                                         01006000
   begin                                                                01006100
      j:=1;                                                             01006200
      k:=2;                                                             01006300
      i:=2                                                              01006400
   end;                                                                 01006500
   if (a[2].i=5) and (a[1].i=2) then                                    01006600
      writeln(' PASS...6.8.3.10-4, WITH')                               01006700
   else                                                                 01006800
      writeln(' FAIL...6.8.3.10-4, WITH');                              01006900
end.                                                                    01007000
```

```
{TEST 6.8.3.10-5, CLASS=CONFORMANCE}                                    01007100
                                                                        01007200
{ This test checks that the selection of a variable in the              01007300
  record-variable-list is performed before the component                01007400
  statement is executed. The compiler fails if the program              01007500
  does not compile or the program prints FAIL. }                        01007600
                                                                        01007700
program t6p8p3p10d5(output);                                            01007800
type                                                                    01007900
   pointer = ↑recordtype;                                               01008000
   recordtype = record                                                  01008100
                  data:integer;                                         01008200
                  link:pointer                                          01008300
                end;                                                    01008400
var                                                                     01008500
   counter:integer;                                                     01008600
   p,q:pointer;                                                         01008700
begin                                                                   01008800
   counter:=0;                                                          01008900
   new(p);                                                              01009000
   p↑.data:=0;                                                          01009100
   new(q);                                                              01009200
   q↑.data:=1;                                                          01009300
   q↑.link:=nil;                                                        01009400
   p↑.link:=q;                                                          01009500
   q:=p;                                                                01009600
   with q↑ do                                                           01009700
   begin                                                                01009800
      q:=link;                                                          01009900
      if (data=0) and (q↑.data=1) then                                  01010000
         counter:=counter+1;                                            01010100
   end;                                                                 01010200
   with p↑ do                                                           01010300
   begin                                                                01010400
      p:=link;                                                          01010500
      { The first record now has no reference, so it could              01010600
        be deleted prematurely. }                                       01010700
      if (data=0) and (p↑.data=1) then                                  01010800
         counter:=counter+1;                                            01010900
   end;                                                                 01011000
   if counter=2 then                                                    01011100
         writeln(' PASS...6.8.3.10-5, WITH')                            01011200
      else                                                              01011300
          writeln(' FAIL...6.8.3.10-5, WITH');                          01011400
end.                                                                    01011500
```

```
{TEST 6.8.3.10-6, CLASS=CONFORMANCE}                                    01011600
                                                                        01011700
{ This test checks that the order of evaluation of the                  01011800
  record-variable-list in a with statement is correctly                 01011900
  implemented. The compiler fails if the program prints FAIL. }         01012000
                                                                        01012100
program t6p8p3p10d6(output);                                            01012200
type                                                                    01012300
   pp = ↑ptr;                                                           01012400
   ptr = record                                                         01012500
           i:integer;                                                   01012600
           link:pp;                                                     01012700
         end;                                                           01012800
var                                                                     01012900
   p,q,r : pp;                                                          01013000
begin                                                                   01013100
   new(p);                                                              01013200
   p↑.i := 0;                                                           01013300
   new(q);                                                              01013400
   q↑.i := 0;                                                           01013500
   p↑.link := q;                                                        01013600
   new(r);                                                              01013700
   r↑.i := 0;                                                           01013800
   r↑.link := nil;                                                      01013900
   q↑.link := r;                                                        01014000
   with p↑, link↑, link↑ do                                             01014100
   begin                                                                01014200
      i:=5;                                                             01014300
   end;                                                                 01014400
   if ((r↑.i=5) and (q↑.i=0) and (p↑.i=0)) then                         01014500
      writeln('PASS...6.8.3.10-6, WITH')                                01014600
   else                                                                 01014700
      writeln(' FAIL...6.8.3.10-6, WITH');                              01014800
end.                                                                    01014900
```

```
{TEST 6.8.3.10-7, CLASS=QUALITY}

{ This test checks that with statements may be nested to 15
  levels. The test may break a compiler limit in some compilers,
  particularly if a register is allocated for every selected
  variable. }

program t6p8p3p10d7(output);
type
   rec1 = record
            i:integer
          end;
   rec2 = record
            i:integer
          end;
   rec3 = record
            i:integer
          end;
   rec4 = record
            i:integer
          end;
   rec5 = record
            i:integer
          end;
   rec6 = record
            i:integer
          end;
   rec7 = record
            i:integer
          end;
   rec8 = record
            i:integer
          end;
   rec9 = record
            i:integer
          end;
   rec10 = record
            i:integer
          end;
   rec11 = record
            i:integer
          end;
   rec12 = record
            i:integer
          end;
   rec13 = record
            i:integer
          end;
   rec14 = record
            i:integer
          end;
   rec15 = record
            i:integer
          end;
   p1 = ↑rec1;
   p2 = ↑rec2;
   p3 = ↑rec3;
   p4 = ↑rec4;
   p5 = ↑rec5;
   p6 = ↑rec6;
   p7 = ↑rec7;
   p8 = ↑rec8;
   p9 = ↑rec9;
   p10 = ↑rec10;
   p11 = ↑rec11;
   p12 = ↑rec12;
   p13 = ↑rec13;
   p14 = ↑rec14;
   p15 = ↑rec15;
var
   ptr1 : p1;
   ptr2 : p2;
   ptr3 : p3;
   ptr4 : p4;
   ptr5 : p5;
   ptr6 : p6;
   ptr7 : p7;
   ptr8 : p8;
   ptr9 : p9;
   ptr10 : p10;
   ptr11 : p11;
   ptr12 : p12;
   ptr13 : p13;
   ptr14 : p14;
   ptr15 : p15;
begin
   new(ptr1); ptr1↑.i:=0;
   new(ptr2); ptr2↑.i:=0;
   new(ptr3); ptr3↑.i:=0;
   new(ptr4); ptr4↑.i:=0;
   new(ptr5); ptr5↑.i:=0;
   new(ptr6); ptr6↑.i:=0;
   new(ptr7); ptr7↑.i:=0;
   new(ptr8); ptr8↑.i:=0;
   new(ptr9); ptr9↑.i:=0;
   new(ptr10); ptr10↑.i:=0;
   new(ptr11); ptr11↑.i:=0;
   new(ptr12); ptr12↑.i:=0;
   new(ptr13); ptr13↑.i:=0;
   new(ptr14); ptr14↑.i:=0;
   new(ptr15); ptr15↑.i:=0;
   with ptr1↑ do
      with ptr2↑ do
         with ptr3↑ do
            with ptr4↑ do
               with ptr5↑ do
                  with ptr6↑ do
                     with ptr7↑ do
                        with ptr8↑ do
                           with ptr9↑ do
                              with ptr10↑ do
                                 with ptr11↑ do
                                    with ptr12↑ do
                                       with ptr13↑ do
                                          with ptr14↑ do
                                             with ptr15↑ do
                                                i:=5;
   writeln(' >15 LEVELS OF WITH STATEMENTS ALLOWED...6.8.3.10-7');
end.
```

```
{TEST 6.9.1-1, CLASS=CONFORMANCE}

{ This program checks that the functions eoln and eof are
  correctly implemented. The compiler fails if the program does
  not compile or the program prints FAIL. }

program t6p9p1d1(f,output);
var
   f:text;
   counter:integer;
   c:char;
begin
   rewrite(f);
   counter:=0;
   writeln(f,1);
   writeln(f,'A');
   reset(f);
   while not eoln(f) do
      read(f,c);
   read(f,c);
   if (c=' ') then
      counter:=counter+1;
   read(f,c);
   if (c='A') then
      counter:=counter+1;
   if eoln(f) then
     counter:=counter+1;
   read(f,c);
   if eof(f) then
      counter:=counter+1;
   if (counter=4) then
      writeln(' PASS...6.9.1-1, EOLN AND EOF')
   else
      writeln('FAIL...6.9.1-1, EOLN AND EOF');
end.
```

```
{TEST 6.9.2-1, CLASS=CONFORMANCE}

{ This test checks that a single read statement with many
  variables is equivalent to many read statements containing
  one variable each. The compiler fails if the program does not
  compile or the program prints FAIL. }

program t6p9p2d1(f,output);
var
   f:text;
   a,b,c,d,e:integer;
   a1,b1,c1,d1,e1:integer;
begin
   rewrite(f);
   writeln(f,' 1 2 3 4 5 ');
   reset(f);
   read(f,a,b,c,d,e);
   reset(f);
   read(f,a1);
   read(f,b1);
   read(f,c1);
   read(f,d1);
   read(f,e1);
   if(a=a1) and (b=b1) and (c=c1) and (d=d1) and (e=e1) then
      writeln(' PASS...6.9.2-1, READ')
   else
      writeln(' FAIL...6.9.2-1, READ');
end.


{TEST 6.9.2-2, CLASS=CONFORMANCE}

{ This test checks that a read of a character variable is
  equivalent to correctly positioning the buffer variable.
  The compiler fails if the program does not compile or the program
  prints FAIL. }

program t6p9p2d2(f,output);
var
   f:text;
   a,b,a1,b1:char;
begin
   rewrite(f);
   writeln(f,'ABC');
   reset(f);
   read(f,a);
   read(f,b);
   reset(f);
   a1:=f↑; get(f);
   b1:=f↑; get(f);
   if(a=a1) and (b=b1) then
      writeln(' PASS...6.9.2-2, READ')
   else
      writeln(' FAIL...6.9.2-2, READ');
end.
```

```
{TEST 6.9.2-3, CLASS=CONFORMANCE}                                    01035700
                                                                     01035800
{ This test checks that integers and reals are read correctly        01035900
  from a file. The compiler fails if the program does not            01036000
  compile or the program prints FAIL. }                              01036100
                                                                     01036200
program t6p9p2d3(f,output);                                          01036300
var                                                                  01036400
   f:text;                                                           01036500
   i,j:integer;                                                      01036600
   r,s:real;                                                         01036700
begin                                                                01036800
                                                                     01036900
   { Internal (compile-time conversions) and run-time conversions    01037000
     should result in the same value, hence justifying the           01037100
     equality tests on real numbers. }                               01037200
                                                                     01037300
   rewrite(f);                                                       01037400
   writeln(f,' 123 123.456 5 123E6 ');                               01037500
   reset(f);                                                         01037600
   read(f,i,r,j,s);                                                  01037700
   if(i=123)and(r=123.456) and (j=5) and (s=123E6) then              01037800
      writeln(' PASS...6.9.2-3, READ')                               01037900
   else                                                              01038000
   begin                                                             01038100
      if (i=123) and (j=5) then                                      01038200
         writeln(' FAIL...6.9.2-3, READ REAL CONVERSIONS')           01038300
      else                                                           01038400
         writeln(' FAIL...6.9.2-3, READ')                            01038500
   end;                                                              01038600
end.                                                                 01038700


{TEST 6.9.2-4, CLASS=ERRORHANDLING}                                  01038800
                                                                     01038900
{ This test checks that an error is produced when an attempt         01039000
  is made to read an integer but the sequence of characters          01039100
  on the input file does not form a valid signed integer. }          01039200
                                                                     01039300
program t6p9p2d4(f,output);                                          01039400
var                                                                  01039500
   f:text;                                                           01039600
                                                                     01039700
   i:integer;                                                        01039800
begin                                                                01039900
   rewrite(f);                                                       01040000
   writeln(f,'ABC123');                                              01040100
   reset(f);                                                         01040200
   read(f,i);       {should cause an error}                          01040300
   writeln(' ERROR NOT DETECTED...6.9.2-4');                         01040400
end.                                                                 01040500
```

```
{TEST 6.9.2-5, CLASS=ERRORHANDLING}                                  01040600
                                                                     01040700
{ This test checks that an error is produced when an attempt         01040800
  is made to read a real but the sequence of characters              01040900
  on the input file does not form a valid real. }                    01041000
                                                                     01041100
program t6p9p2d5(f,output);                                          01041200
var                                                                  01041300
   f:text;                                                           01041400
   r:real;                                                           01041500
begin                                                                01041600
   rewrite(f);                                                       01041700
   writeln(f,'ABC123.456');                                          01041800
   reset(f);                                                         01041900
   read(f,r);       {should cause an error}                          01042000
   writeln(' ERROR NOT DETECTED...6.9.2-5');                         01042100
end.                                                                 01042200


{TEST 6.9.3-1, CLASS=CONFORMANCE}                                    01042300
                                                                     01042400
{ This test checks that readln is correctly implemented.             01042500
  The compiler fails if the program does not compile or the          01042600
  program prints FAIL. }                                             01042700
                                                                     01042800
program t6p9p3d1(output);                                            01042900
var                                                                  01043000
   f:text;                                                           01043100
   a,b,c:char;                                                       01043200
   counter:integer;                                                  01043300
begin                                                                01043400
   counter:=0;                                                       01043500
   rewrite(f);                                                       01043600
   writeln(f,'ABC');                                                 01043700
   writeln(f,'DE');                                                  01043800
   reset(f);                                                         01043900
   readln(f,a,b,c);                                                  01044000
   read(f,a);                                                        01044100
   if (a='D') then counter:=counter+1;                               01044200
   reset(f);                                                         01044300
   read(f,a,b,c);                                                    01044400
   readln(f);                                                        01044500
   read(f,a);                                                        01044600
   if(a='D') then counter:=counter+1;                                01044700
   reset(f);                                                         01044800
   read(f,a);                                                        01044900
   while not eoln(f) do get(f);                                      01045000
   get(f);                                                           01045100
   if (f↑='D') then counter:=counter+1;                              01045200
   if (counter=3) then                                               01045300
      writeln(' PASS...6.9.3-1, READLN')                             01045400
   else                                                              01045500
      writeln(' FAIL...6.9.3-1, READLN');                            01045600
end.                                                                 01045700
```

```
{TEST 6.9.4-1, CLASS=CONFORMANCE}                                        01045800
                                                                         01045900
{ This test checks that a write procedure with many parameters           01046000
  is equivalent to many write procedures with one parameter each.        01046100
  The compiler fails if the program does not compile or the program      01046200
  prints FAIL. }                                                         01046300
                                                                         01046400
program t6p9p4d1(f,output);                                              01046500
var                                                                      01046600
   f:text;                                                               01046700
   a,b,c,d,e:char;                                                       01046800
   a1,b1,c1,d1,e1:char;                                                  01046900
   counter:integer;                                                      01047000
begin                                                                    01047100
   counter:=0;                                                           01047200
   rewrite(f);                                                           01047300
   a:='A';                                                               01047400
   b:='B';                                                               01047500
   c:='C';                                                               01047600
   d:='D';                                                               01047700
   e:='E';                                                               01047800
   write(f,a,b,c,d,e);                                                   01047900
   writeln(f);                                                           01048000
   reset(f);                                                             01048100
   read(f,a1,b1,c1,d1,e1);                                               01048200
   if (a=a1) and (b=b1) and (c=c1) and (d=d1) and (e=e1) then            01048300
      counter:=counter+1;                                                01048400
   rewrite(f);                                                           01048500
   write(f,a);                                                           01048600
   write(f,b);                                                           01048700
   write(f,c);                                                           01048800
   write(f,d);                                                           01048900
   write(f,e);                                                           01049000
   writeln(f);                                                           01049100
   reset(f);                                                             01049200
   read(f,a1,b1,c1,d1,e1);                                               01049300
   if(a1=a) and (b1=b) and (c1=c) and (d1=d) and (e1=e) then             01049400
      counter:=counter+1;                                                01049500
   if (counter=2) then                                                   01049600
      writeln(' PASS...6.9.4-1, WRITE')                                  01049700
   else                                                                  01049800
      writeln(' FAIL...6.9.4-1, WRITE');                                 01049900
end.                                                                     01050000
```

```
{TEST 6.9.4-2, CLASS=CONFORMANCE}                                        01050100
                                                                         01050200
{ This test checks that the default value for the field width of a       01050300
  character type is one. The compiler fails if the program does not      01050400
  compile or the program prints FAIL. }                                  01050500
                                                                         01050600
                                                                         01050700
program t6p9p4d2(f,output);                                              01050800
var                                                                      01050900
   f:text;                                                               01051000
   a,b:char;                                                             01051100
begin                                                                    01051200
   rewrite(f);                                                           01051300
   a:='A';                                                               01051400
   b:='B';                                                               01051500
   writeln(f,a,b);                                                       01051600
   reset(f);                                                             01051700
   read(f,a,b);                                                          01051800
   if (a='A') and (b='B') then                                           01051900
      writeln(' PASS...6.9.4-2, WRITE')                                  01052000
   else                                                                  01052100
      write(' FAIL...6.9.4-2, WRITE');                                   01052200
end.                                                                     01052300
```

```
{TEST 6.9.4-3, CLASS=CONFORMANCE}                                        01052400
                                                                         01052500
{ This test checks the implementation of integer output.                 01052600
  The compiler fails if the program does not compile or the program      01052700
  prints FAIL. }                                                         01052800
                                                                         01052900
program t6p9p4d3(f,output);                                              01053000
var                                                                      01053100
   f:text;                                                               01053200
   a:char;                                                               01053300
   b:packed array [1..26] of char;                                       01053400
   i:integer;                                                            01053500
begin                                                                    01053600
   rewrite(f);                                                           01053700
   writeln(f,0:3,1:3,-1:3,10:3,99:3,100:3,-100:3,1111:3);                01053800
   reset(f);                                                             01053900
   for i:=1 to 26 do                                                     01054000
      read(f,b[i]);                                                      01054100
   if (b='  0  1 -1 10 99100-1001111') then                              01054200
      writeln(' PASS...6.9.4-3, WRITE INTEGERS')                         01054300
   else                                                                  01054400
      writeln(' FAIL...6.9.4-3, WRITE INTEGERS');                        01054500
end.                                                                     01054600
```

```
{TEST 6.9.4-4, CLASS=CONFORMANCE}                                          01054700
                                                                           01054800
{ This program checks that real numbers are correctly written to           01054900
  text files. The compiler fails if the program does not compile           01055000
  or the program prints FAIL. }                                            01055100
                                                                           01055200
program t6p9p4d4(f,output);                                                01055300
var                                                                        01055400
   f:text;                                                                 01055500
   a:packed array [1..26] of char;                                         01055600
   b:packed array [1..24] of char;                                         01055700
   i:integer;                                                              01055800
   counter:integer;                                                        01055900
begin                                                                      01056000
   rewrite(f);                                                             01056100
   counter:=0;                                                             01056200
   writeln(f,0.0:6,1.0:6,1.0:10);                                          01056300
   reset(f);                                                               01056400
   for i:=1 to 26 do                                                       01056500
      read(f,a[i]);                                                        01056600
   if (a='     0.0     1.0 1.000E+00') then                                01056700
      counter:=counter+1;                                                  01056800
   rewrite(f);                                                             01056900
   writeln(f,0.0:4:1,1.0:6:1,-1.0:6:1,123.456:7:3);                        01057000
   reset(f);                                                               01057100
   for i:=1 to 24 do                                                       01057200
      read(f,b[i]);                                                        01057300
   if (b=' 0.0   1.0  -1.0 123.456') then                                  01057400
      counter:=counter+1;                                                  01057500
   if (counter=2) then                                                     01057600
      writeln(' PASS...6.9.4-4, WRITE REALS')                              01057700
   else                                                                    01057800
      writeln(' FAIL...6.9.4-4, WRITE REALS');                             01057900
end.                                                                       01058000
```

```
{TEST 6.9.4-5, CLASS=IMPLEMENTATIONDEFINED}                                01058100
                                                                           01058200
{ This program determines the implementation defines value which           01058300
  represents the number of digit characters written in an exponent. }      01058400
                                                                           01058500
program t6p9p4d5(f,output);                                                01058600
var                                                                        01058700
   f:text;                                                                 01058800
   c:char;                                                                 01058900
   i:integer;                                                              01059000
begin                                                                      01059100
   rewrite(f);                                                             01059200
   writeln(f,1.0:10,'ABC');                                                01059300
   reset(f);                                                               01059400
   repeat                                                                  01059500
      read(f,c);                                                           01059600
   until (c='E');                                                          01059700
   read(f,c);                                                              01059800
   i:=-1;                                                                  01059900
   repeat                                                                  01060000
      read(f,c);                                                           01060100
      i:=i+1;                                                              01060200
   until (c='A');                                                          01060300
   writeln(' THE NUMBER OF DIGITS WRITTEN IN AN EXPONENT IS',i:5);         01060400
end.                                                                       01060500
```

```
{TEST 6.9.4-6, CLASS=CONFORMANCE}                                    01060600
                                                                     01060700
{ This test checks that strings are correctly written onto a text    01060800
  file. The compiler fails if the program does not compile or        01060900
  the program prints FAIL. }                                         01061000
                                                                     01061100
program t6p9p4d6(f,output);                                          01061200
var                                                                  01061300
f:text;                                                              01061400
i,j,k,counter:integer;                                               01061500
c:char;                                                              01061600
begin                                                                01061700
   rewrite(f);                                                       01061800
   counter:=0;                                                       01061900
   for i := 1 to 10 do                                               01062000
      writeln(f,'AAAAA':i,'B':1);                                    01062100
   writeln(f,'BBBBB','C':1);                                         01062200
   reset(f);                                                         01062300
   for i:=1 to 10 do                                                 01062400
   begin                                                             01062500
      for j:=6 to i do begin                                         01062600
         read(f,c);                                                  01062700
         if (c=' ') then                                             01062800
            counter:=counter+1;                                      01062900
      end;                                                           01063000
      if (i>5) then k:=5 else k:=i;                                  01063100
      for j:=1 to k do                                               01063200
      begin                                                          01063300
         read(f,c);                                                  01063400
         if(c='A') then                                              01063500
            counter:=counter+1;                                      01063600
      end;                                                           01063700
      read(f,c);                                                     01063800
      if (c='B') then                                                01063900
         counter:=counter+1;                                         01064000
      readln(f);                                                     01064100
   end;                                                              01064200
   for i:=1 to 5 do                                                  01064300
   begin                                                             01064400
      read(f,c);                                                     01064500
      if (c='B') then                                                01064600
         counter:=counter+1;                                         01064700
   end;                                                              01064800
   read(f,c);                                                        01064900
   if (c='C') then                                                   01065000
      counter:=counter+1;                                            01065100
   if(counter=71) then                                               01065200
      writeln(' PASS...6.9.4-6, WRITE STRINGS')                      01065300
   else                                                              01065400
      writeln(' FAIL...6.9.4-6, WRITE STRINGS');                     01065500
end.                                                                 01065600
```

```
{TEST 6.9.4-7, CLASS=CONFORMANCE}                                    01065700
                                                                     01065800
{ This test checks that boolean variables are correctly written      01065900
  to text files. The compiler fails if the program does not compile  01066000
  or the program prints FAIL. }                                      01066100
                                                                     01066200
program t6p9p4d7(f,output);                                          01066300
var                                                                  01066400
   f:text;                                                           01066500
   b,c:boolean;                                                      01066600
   a:packed array[1..10] of char;                                    01066700
   i:integer;                                                        01066800
begin                                                                01066900
   { This treatement is believed to be very dubious and may be       01067000
     altered in the future versions of the standard:                 01067100
     A.H.J. Sale 1979 June 1 }                                       01067200
   rewrite(f);                                                       01067300
   b:=true;                                                          01067400
   c:=not b;                                                         01067500
   writeln(f,b:5,c:5);                                               01067600
   reset(f);                                                         01067700
   for i:=1 to 10 do                                                 01067800
      read(f,a[i]);                                                  01067900
   if (a='TRUE FALSE') then                                          01068000
      writeln(' PASS...6.9.4-7, WRITE BOOLEAN')                      01068100
   else                                                              01068200
      writeln(' FAIL...6.9.4-7, WRITE BOOLEAN');                     01068300
end.                                                                 01068400


{TEST 6.9.4-8, CLASS=DEVIANCE}                                       01068500
                                                                     01068600
{ This program attempts to output an integer number using a real     01068700
  format. The compiler deviates if the program prints DEVIATES. }    01068800
                                                                     01068900
program t6p9p4d8(output);                                            01069000
var                                                                  01069100
   i:integer;                                                        01069200
begin                                                                01069300
   i:=123;                                                           01069400
   writeln(i:6:1);                                                   01069500
   writeln(' DEVIATES...6.9.4-8, WRITE');                            01069600
end.                                                                 01069700


{TEST 6.9.4-9, CLASS=DEVIANCE}                                       01069800
                                                                     01069900
{ This test attempts to output integers whose field width parameter  01070000
  are zero or negative. The compiles deviates if the program prints  01070100
  DEVIATES. }                                                        01070200
                                                                     01070300
program t6p9p4d9(output);                                            01070400
var                                                                  01070500
   i:integer;                                                        01070600
begin                                                                01070700
   for i:=10 downto -1 do                                            01070800
      writeln(' ','.':i, 'REP=',i);                                  01070900
   writeln(' DEVIATES...6.9.4-9, WRITE');                            01071000
end.                                                                 01071100
```

```
{TEST 6.9.4-10, CLA  QUALITY}                                          01071200
                                                                       01071300
{ This program checks that data written appears on the output          01071400
  file regardless of the omission of a line marker. The common         01071500
  error is to buffer output and fail to flush the buffers at           01071600
  end of job. }                                                        01071700
                                                                       01071800
program t6p9p4d10(output);                                             01071900
begin                                                                  01072000
   write(' OUTPUT IS FLUSHED AT END_OF_JOB...6.9.4-10')                01072100
end.                                                                   01072200
```

```
{TEST 6.9.4-11, CLASS=IMPLEMENTATIONDEFINED}                           01072300
                                                                       01072400
{ This program determines the implementation defined default           01072500
  field width for writing integer, boolean and real types. }           01072600
                                                                       01072700
program t6p9p4d11(f,output);                                           01072800
var                                                                    01072900
   f:text;                                                             01073000
   c:char;                                                             01073100
   i,j:integer;                                                        01073200
                                                                       01073300
   function readfield:integer;                                         01073400
   var                                                                 01073500
      i:integer;                                                       01073600
   begin                                                               01073700
   i:=0;                                                               01073800
   repeat                                                              01073900
      read(f,c);                                                       01074000
      i:=i+1;                                                          01074100
   until (c='Z');                                                      01074200
   readfield:=i-1;                                                     01074300
end;                                                                   01074400
                                                                       01074500
begin                                                                  01074600
   rewrite(f);                                                         01074700
   writeln(f,1,'Z',100,'Z');                                           01074800
   writeln(f,false,'Z',true,'Z');                                      01074900
   writeln(f,1.0,'Z',1000.0,'Z');                                      01075000
   reset(f);                                                           01075100
   writeln(' IMPLEMENTATION DEFINED DEFAULT FIELD WIDTH VALUES');      01075200
   i:=readfield;                                                       01075300
   j:=readfield;                                                       01075400
   if (i=j) then                                                       01075500
      writeln(' INTEGERS:',i:5,' CHARACTERS')                          01075600
   else                                                                01075700
   writeln(' THE VALUE VARIES ACCORDING TO THE SIZE OF THE INTEGER');   01075800
   readln(f);                                                          01075900
   i:=readfield;                                                       01076000
   j:=readfield;                                                       01076100
   if (i=j) then                                                       01076200
      writeln(' BOOLEAN:',i:5,' CHARACTERS')                           01076300
   else                                                                01076400
      writeln(' THE VALUE VARIES ACCORDING TO THE BOOLEAN VALUE');      01076500
   readln(f);                                                          01076600
   i:=readfield;                                                       01076700
   j:=readfield;                                                       01076800
   if (i=j) then                                                       01076900
      writeln(' REAL:',i:5,' CHARACTERS')                              01077000
   else                                                                01077100
      writeln(' THE VALUE VARIES ACCORDING TO THE SIZE OF THE REAL');   01077200
end.                                                                   01077300
```

```
{TEST 6.9.4-12, CLASS=DEVIANCE}

{This program checks whether an unpacked array of characters
  can be output. The compiler deviates if the program prints
  DEVIATES. }

program t6p9p4d12(output);
var
   s:array[1..3] of char;
begin
   s[1]:='R'; s[2]:='A'; s[3]:='N';
   writeln(' RAN=',s);
   writeln(' DEVIATES...6.9.4-12, WRITE');
end.


{TEST 6.9.4-13, CLASS=CONFORMANCE}

{ This program attempts to perform recursive I/O using a
  different file for the second I/O action. }

program t6p9p4d13(f,output);
var
   f:text;

function a(i:integer):integer;
begin
   writeln(f,i);
   a:=i;
end;

begin
   rewrite(f);
   writeln(a(1));
   writeln(' RECURSIVE I/O ALLOWED USING DIFFERENT FILES');
   writeln(' PASS...6.9.4-13, RECURSIVE I/O');
end.


{TEST 6.9.4-14, CLASS=QUALITY}

{ This program attempts to perform recursive I/O using the
  same file for the second I/O action.
  The semantics of write are not sufficiently well-defined to
  establish what should occur. It depends on evaluation orders,
  etc., which is why this test is in the quality section. }

program t6p9p4d14(f,output);

function a(i:integer):integer;
begin
   writeln(i);
   a:=i;
end;

begin
   writeln(a(1));
   writeln('RECURSIVE I/O ALLOWED USING THE SAME FILE...6.9.4-14');
end.
```

```
{TEST 6.9.4-15, CLASS=CONFORMANCE}

{ This test checks that a write that does not specify the file
  always writes on the default file at the program level, not
  any local variable with the same name. }

program t6p9p4d15(output);
   procedure p;
   var
      output:text;
   begin
      rewrite(output);
      writeln(output,' FAIL...6.9.4-15');
      writeln(' PASS...6.9.4-15')
   end;
begin
   p
end.


{TEST 6.9.5-1, CLASS=CONFORMANCE}

{ This program checks the implementation of procedure writeln.
  The compiler fails if the program prints FAIL or the program
  does not compile. }

program t6p9p5d1(f,output);
var
   f:text;
   a,b:packed array[1..10] of char;
   i:integer;
begin
   rewrite(f);
   writeln(f,1:5,'ABCDE');
   write(f,1:5,'ABCDE');
   writeln(f);
   reset(f);
   for i:=1 to 10 do
      read(f,a[i]);
   reset(f);
   for i:=1 to 10 do
      read(f,b[i]);
   if (a=b) then
      writeln(' PASS...6.9.5-1, WRITELN')
   else
      writeln(' FAIL...6.9.5-1, WRITELN');
end.
```

```
{TEST 6.9.6-1, CLAS    NFORMANCE}                                        01087400
                                                                         01087500
{ This program checks that the procedure page is implemented.           01087600
  This conformance test is unable to determine whether the compiler      01087700
  passes or fails - the user must check that a page has been             01087800
  generated. }                                                           01087900
                                                                         01088000
program t6p9p6d1(output);                                                01088100
begin                                                                    01088200
   writeln(' PAGE GENERATION TEST');                                     01088300
   page(output);                                                         01088400
   writeln(' IF THIS LINE IS PRINTED ON THE TOP OF A NEW PAGE');         01088500
   writeln(' THEN PASS...6.9.6-1, PAGE');                                01088600
   writeln(' ELSE FAIL...6.9.6-1, PAGE');                                01088700
end.                                                                     01088800


{TEST 6.10-1, CLASS=DEVIANCE}                                            01088900
                                                                         01089000
{ This test checks the effect of using a default file not declared in    01089100
  the program heading. The compiler deviates if the program              01089200
  prints DEVIATES. }                                                     01089300
                                                                         01089400
program t6p10d1(input);                                                  01089500
begin                                                                    01089600
   writeln(' DEVIATES...6.10-1, FILE DECLARATION');                      01089700
end.                                                                     01089800


{TEST 6.10-2, CLASS=IMPLEMENTATIONDEFINED}                               01089900
                                                                         01090000
{ This program checks the effect of doing a rewrite on the               01090100
  standard file output. The effect is implementation dependent. }        01090200
                                                                         01090300
program t6p10d2(output);                                                 01090400
begin                                                                    01090500
   rewrite(output);                                                      01090600
   writeln(' IMPLEMENTATION DEPENDENT...6.10-2');                        01090700
   writeln(' A REWRITE HAS BEEN PERFORMED ON FILE OUTPUT');              01090800
end.                                                                     01090900


{TEST 6.10-3, CLASS=DEVIANCE}                                            01091000
                                                                         01091100
{ This program checks that the default file output is                    01091200
  implicitly declared at the program level by attempting to              01091300
  redefine it. The file input should be identical, of course.            01091400
  The test should not compile. }                                         01091500
                                                                         01091600
program t6p10d3(output);                                                 01091700
var                                                                      01091800
   output:integer;                                                       01091900
begin                                                                    01092000
   output:=1;                                                            01092100
   writeln(' DEVIATES...6.10-3, OUTPUT REDEFINED')                       01092200
end.                                                                     01092300
```

```
{TEST 6.10-4, CLASS=DEVIANCE}                                            01092400
                                                                         01092500
{ This program has no program statement. Some compilers may              01092600
  assume the existence of such a statement if none is present.           01092700
  The compiler deviates if the program compiles and prints               01092800
  DEVIATES. }                                                            01092900
                                                                         01093000
var                                                                      01093100
   i:integer;                                                            01093200
begin                                                                    01093300
   i:=5;                                                                 01093400
   writeln(' DEVIATES...6.10-4, PROGRAM')                                01093500
end.                                                                     01093600


{TEST 6.11-1, CLASS=IMPLEMENTATIONDEFINED}                               01093700
                                                                         01093800
{ This program checks whether equivalent symbols can be used for         01093900
  the standard reference representation. The equivalent symbols          01094000
  for comment delimiters are tested. They are implemented if             01094100
  the program prints ALTERNATE COMMENT DELIMITERS IMPLEMENTED. }         01094200
                                                                         01094300
program t6p11d1(output);                                                 01094400
(* Test of alternate comment delimiters *)                              01094500
begin                                                                    01094600
 (* test of alternate comment delimiters. If these delimiters            01094700
   are not implemented the compiler will give a syntax error. *)         01094800
   writeln(' ALTERNATE COMMENT DELIMITERS IMPLEMENTED...6.11-1');        01094900
end.                                                                     01095000
```

```
{TEST 6.11-2, CLASS=IMPLEMENTATIONDEFINED}                          01095100
                                                                    01095200
{ This program checks whether equivalent symbols can be used for    01095300
  the standard reference representation. The equivalent symbols     01095400
  for the up-arrow, :, ;, :=, and [ ] are tested. The equivalent    01095500
  symbols are implemented if the program prints                     01095600
  EQUIVALENT SYMBOLS ARE IMPLEMENTED. }                             01095700
                                                                    01095800
program t6p11d2(output);                                            01095900
type                                                                01096000
   rec = record                                                     01096100
           a,b:integer                                              01096200
         end;                                                       01096300
   ptr1=@rec;                                                       01096400
   ptr2=↑rec;                                                       01096500
                                                                    01096600
   { The above two statements use the equivalent symbols for the    01096700
     up-arrow }                                                     01096800
                                                                    01096900
var                                                                 01097000
   arr:array (. 1..10 .) of integer;                                01097100
   i % integer;                                                     01097200
   r:real;                                                          01097300
   s:real;                                                          01097400
   j,k,l : integer;                                                 01097500
begin                                                               01097600
   j .= 5;                                                          01097700
   k %= 6;                                                          01097800
   s:=1.0;                                                          01097900
   l := 7  .,                                                       01098000
   s:=1.0;                                                          01098100
   writeln(' EQUIVALENT SYMBOLS ARE IMPLEMENTED...6.11-2');         01098200
end.                                                                01098300


{TEST 6.11-3, CLASS=IMPLEMENTATIONDEFINED}                          01098400
                                                                    01098500
{ This program checks whether equivalent symbols can be used for    01098600
  the standard reference representation. The equivalent symbols     01098700
  for the arithmetic operators <,>,<=,>=,<> are checked. They       01098800
  are implemented if the program prints EQUIVALENT SYMBOLS ARE      01098900
  IMPLEMENTED. }                                                    01099000
                                                                    01099100
program t6p11d3(output);                                            01099200
var                                                                 01099300
   j,k,l : integer;                                                 01099400
   b : boolean;                                                     01099500
begin                                                               01099600
   j:=1;                                                            01099700
   k:=2;                                                            01099800
   b := j GT k;                                                     01099900
   b := j LT k;                                                     01100000
   b := j GE k;                                                     01100100
   b := j LE k;                                                     01100200
   b := j NE k;                                                     01100300
   writeln(' EQUIVALENT SYMBOLS ARE IMPLEMENTED...6.11-3');         01100400
end.                                                                01100500
```

THREE SAMPLE VALIDATION REPORTS

{ A fourth report came to hand very late in the preparation of this issue, and it is also included. }

Introduction

In this section we present three samples of Validation Reports on processors. Care is needed in the interpretation of these Reports for several reasons. Firstly, they are a snapshot in time of the processor concerned; some of the reported flaws will be fixed by the maintainers, perhaps even before this is printed. Secondly, some processors contain intended extensions, or have an interpretation which anticipates a change in the draft Standard in its route to finalization. Nevertheless, it is felt that publication of these Reports will

(a) encourage other users to test processors accessible to them and publish the results in Pascal News,

(b) indicate likely portability problem areas, and

(c) illustrate the type of report which will be meaningful to users of Pascal.

It must be emphasized that these reports are simply of processors which were reasonably convenient for us to test. The report on the Burroughs B6700 compiler originating at the University of California is a user's viewpoint of an unmodified system. There is no good reason to suspect that the report is particularly bad (or particularly good). It is likely to be representative of the results other users will achieve with their processors.

The report on the Burroughs B6700/7700 compiler originating at the University of Tasmania is somewhat different. Firstly, it is our own processor, and the report is therefore prepared with greater knowledge of what is happening (not simply noting a bald failure for unknown reasons as we have had to record in other situations). Secondly, and more importantly, it has been the prime testbed for the validation suite which has resulted in most of the minor faults being fixed as soon as they are detected. Consequently, only a few of the more difficult areas remain to be reported by the Validation Suite.

The report of the P4 compiler gives an indication of how the portable Pascal systems conform to the standard.

The test runs which led to these reports were carried out by:

R.A. Freak
C.D. Keen

Annotation and analysis were carried out by:

R.A. Freak
A.H.J. Sale
C.D. Keen

(For the benefit of Pascal users who want to carry out similar Validation Tests on their processors, we estimate the time required to do this as about 1-5 man days, depending on familiarity with the processor, turn-around time, etc. Fixing the flaws, of course, takes a lot longer).

B6700 - UC

PASCAL VALIDATION SUITE REPORT

Pascal Processor Identification

Computer: Burroughs B6700

Processor: B6700 Pascal version 2.9.178.008
(University of California at San Diego compiler)

Test Conditions

Tester: R.A. Freak (a user at the University of Tasmania)

Date: August 1979

Validation Suite Version: 2.2

Conformance Tests

Number of tests passed: 118

Number of tests failed: 21 (13 basic causes)

Details of failed tests:

Test 6.1.5-2 failed because long numbers are not accepted by the compiler.

Tests 6.4.3.2-3, 6.6.3.1-2 and 6.8.2.1-1 failed because booleans are not permitted to be used as array indexes.

Tests 6.4.3.3-1 and 6.4.3.3-3 fail because empty records or empty fields are not permitted by the compiler.

Test 6.4.3.5-1 shows that a file of pointer is not allowed.

Tests 6.4.3.5-2, 6.6.5.2-3, 6.6.5.2-4, 6.6.5.2-5, 6.9.1-1 all fail because the implementation of textfiles is non standard - particularly the handling of *eof* and *eoln*.

Test 6.4.3.5-4 indicates that the file output is not flushed at the end of a program.

Test 6.5.1-1 shows that a *file of char* is not permitted in a record.

Test 6.6.3.1-5 failed because of an inconsistency in the implementation of procedure parameters.

Test 6.6.5.4-1 failed because the procedures pack and unpack have not been implemented according to the standard.

Test 6.8.3.5-4 fails because the maximum range of case labels is 1000.

Tests 6.9.4-4 and 6.9.4-7 fail because I/O involving reals and booleans is not standard.

Test 6.9.4-13 fails because a file variable is expected in a write statement.

Test 6.9.4-15 fails because a local file *output* is used for the default file.

Deviance Tests

Number of deviations correctly detected: 62

Number of tests showing true extensions: 5

Number of tests not detecting erroneous deviations: 28 (11 basic causes)

Details of extensions:

Test 6.1.5-6 shows that a lower case e may be used in real numbers (e.g. 1.602e-20).

Test 6.1.7-6 shows that strings can have bounds other than 1..n.

Test 6.1.7-11 shows that a null string is accepted by the compiler.

Tests 6.8.3.9-9 and 6.8.3.9-14 indicate that a non-local variable or a global variable may be used for a for statement control variable.

Details of deviations not detected:

Test 6.1.2-1 shows that the reserved word nil may be redefined.

Tests 6.2.2-4, 6.2.2-9, 6.3-5, 6.3-6, 6.4.1-3 contain a scope error which is not detected by the compiler.

Tests 6.4.5-2, 6.4.5-3, 6.4.5-4, 6.4.5-5 and 6.4.5-13 indicate that type compatibility is used with var parameters rather than enforcing identical types.

Test 6.6.1-6 shows that a forward procedure declaration without a procedure body is not detected.

Test 6.6.3.5 shows that a function without an assignment to the function value variable in its block compiles and runs.

Tests 6.6.3.3-2, 6.6.3.6-2 and 6.6.3.6-4 show that function parameters are assignment compatible.

Test 6.6.6.5-3 shows that the function *odd* may be used with a real parameter.

Test 6.7.2.2-9 shows that a unary plus sign may be used with character operands.

Tests 6.8.2.4-2, 6.8.2.4-3 and 6.8.2.4-4 show that a goto between branches of a statement is permitted.

Tests 6.8.3.8-2, 6.8.3.8-3, 6.8.3.8-4, 6.8.3.8-5, 6.8.3.8-16 and 6.8.3.8-19 show that a for statement control variable may be altered during the execution of the for loop.

Test 6.9.4-9 indicates that a negative field width may be used in a write statement.

Error handling

Number of errors correctly detected: 18

Number of errors not detected: 27 (12 basic causes)

Details of errors not detected

Tests 6.4.3.3-5, 6.4.3.3-6, 6.4.3.3-7 and 6.4.3.3-8 indicate that no checking is performed on the tag field of variant records.

Test 6.4.3.3-12 shows that an assignment to an empty record is not detected.

Test 6.4.6-4 indicates that no bounds checking is performed on subranges.

Tests 6.4.6-7 and 6.4.6-8 indicate that no bounds checking is performed on set operations.

Test 6.6.2-6 shows that the use of a function without an assignment to the function-value-variable is permitted.

Tests 6.6.5.2-1, 6.6.5.2-2, 6.6.5.2-6 and 6.6.5.2-7 fail because I/O has not been implemented strictly according to the standard - particularly the handling of *eof* and *eoln*.

Tests 6.6.5.3-3, 6.6.5.3-5 and 6.6.5.3-6 fail because no check is performed on the pointer parameter of dispose.

Tests 6.6.5.3-7, 6.6.5.3-8 and 6.6.5.3-9 fail because no checks are inserted to check pointers after they have been assigned a value using the variant form of new.

Tests 6.6.6.4-4, 6.6.6.4-5 and 6.6.6.4-7 fail because no bounds checks are inserted for the *succ*, *pred* or *chr* functions.

Tests 6.7.2.2-6 and 6.7.2.2-7 fail because integer overflow/underflow is not detected.

Test 6.7.2.4-1 fails because operations on overlapping sets are not detected.

Tests 6.8.3.9-6 and 6.8.3.9-17 fail because a for control variable is not invalid after the execution of the for loop.

Implementationdefined

Number of tests run: 15

Number of tests incorrectly handled: 1

Details of implementation-dependence

Test 6.4.2.2-7 shows maxint to be 549755813887.

Tests 6.4.3.4-2 and 6.4.3.4-4 show that all set bounds must be positive. A set of char is permitted.

Test 6.6.6.1-1 shows that no standard functions may be used as parameters.

Test 6.6.6.2-11 details some machine characteristics regarding number formats.

Tests 6.7.2.3-2 and 6.7.2.3-3 show that boolean expressions are fully evaluated.

Tests 6.8.2.2-1 and 6.8.2.2-2 show that a variable is selected before the expression is evaluated in an assignment statement.

Test 6.9.4-11 details the default field width specifications: 10 for integers and booleans. The output format for reals fails in test 6.9.4-5.

Test 6.10-2 indicates that a rewrite on the standard file output is permissible.

Tests 6.11-1, 6.11-2 and 6.11-3 show that the alternative comment delimiters have been implemented, as have the alternative pointer symbols. No other equivalent symbols have been implemented.

Quality Measurement

Number of tests run: 23

Number of tests incorrectly handled: 0

Results of tests:

Test 5.2.2-1 shows that identifiers are not distinguished over their whole length; only the first 12 characters are used.

Test 6.1.3-3 shows the number of significant characters in an identifier to be 12.

Test 6.1.8-4 shows that no warning is given if a valid statement or a semicolon is detected in a comment.

Tests 6.2.1-8, 6.2.1-9 and 6.5.1-2 indicate that large lists of declarations may be made in each block.

An array with an integer indextype is not permitted (test 6.4.3.2-4).

Test 6.4.3.3-9 shows that variant fields of a record occupy the same space, using the declared order.

Test 6.4.3.4-5 (Warshall's algorithm) took 10.497744 seconds CPU time and 169 bytes on the Burroughs B6700.

Test 6.6.1-7 shows that procedures cannot be nested to a level greater than 8.

Tests 6.6.6.2-6, 6.6.6.2-7, 6.6.6.2-8, 6.6.6.2-9 and 6.6.6.2-10 tested the sqrt, atan, exp, sin/cos and ln functions and all tests were completed successfully, without any significant errors in the values.

Test 6.7.2.2-4 shows that div and mod have been implemented consistently. mod returns the remainder of div.

Test 6.8.3.5-2 shows that case constants do not have to be of the same type as the case-index, if the case-index is a subrange, but the constants must be compatible with the case-index.

Test 6.8.3.5-8 shows that a large case statement (>256 selections) is permissible.

Test 6.8.3.9-18 indicates that range checking is always used in a case statement after a for statement to check the for variable.

Tests 6.8.3.9-20 and 6.8.3.10-7 indicate that for and with statements may be nested to a depth greater than 15.

Test 6.9.4-10 shows that file buffers are flushed at the end of the program.

Test 6.9.4-14 indicates that recursive I/O is permitted, using the same file.

Extensions

Number of tests run: 1

Details of test:

Test 6.8.3.5-14 shows that the *otherwise* clause in a case statement has not been implemented. However, a construct using an ELSE: label has been implemented.

B6700 – Tas

PASCAL VALIDATION SUITE REPORT

Pascal Processor Identification

Computer: Burroughs B6700

Processor: B6700 Pascal version 2.9.001
(University of Tasmania compiler)

Test Conditions

Tester: R.A. Freak (implementation/maintenance team member)

Date: August 1979

Validation Suite Version: 2.2

Conformance Tests

Number of tests passed: 117

Number of tests failed: 22

Details of failed tests:

Most of the failed tests fall into two categories - the B6700 Pascal I/O is non-standard and the passing of procedure/function parameters has not been implemented.

Tests 6.4.3.1-3, 6.4.3.5-1, 6.4.3.5-2, 6.4.3.5-3, 6.5.1-1, 6.5.3.4-1, 6.6.5.2-3, 6.6.5.2-4, 6.6.5.2-5, 6.9.1-1, 6.9.2-2, 6.9.3-1, 6.9.4-3, 6.9.4-4, 6.9.4-7, 6.9.4-15, all fail because *text* has not been predefined, or the *eof* action or output format is not strictly standard-conforming.

Tests 6.6.3.1-3, 6.6.3.1-5, 6.6.3.4-1, 6.6.3.4-2 and 6.6.3.5-1 fail because the passing of procedure/function parameters has not yet been implemented.

Test 6.4.3.3-1 fails because an empty record containing a semi-colon produces a syntax error.

Deviance Test

Number of deviations correctly detected: 71

Number of tests showing true extensions: 5

Number of tests not detecting erroneous deviations: 13 (5 basic causes)

Number of tests failed: 6 (2 basic causes)

Details of extensions:

Test 6.1.5-6 shows that the lower case e may be used in real numbers (for example 1.602e-20).

Test 6.6.2-5 shows that a function without an assignment to the function variable in its block compiles - the error is detected at run time as an uninitialized value.

Test 6.9.3-8 shows that integers may be written using real formats.

Test 6.10-1 shows that the file parameters in the program heading are ignored in B6700 Pascal.

Test 6.10-3 shows that the file output may be redefined at the program level.

Details of deviations not detected:

Test 6.1.2-1 shows that *nil* may be redefined.

Tests 6.2.2-4, 6.3-6 and 6.4.1-3 show that a common scope error was not detected by the compiler.

Tests 6.4.5-2, 6.4.5-3, 6.4.5-4, 6.4.5-5 and 6.4.5-13 indicate that type compatibility is used with var parameter elements rather than enforcing identical types.

Tests 6.8.2.4-2, 6.8.2.4-3 and 6.8.2.4-4 show that a goto between branches of a statement is permitted.

Test 6.9.4-9 shows that integers may be written even though the field width is too small, but the format used is non-standard.

Details of failed tests:

Tests 6.6.3.5-2, 6.6.3.6-2, 6.6.3.6-3, 6.6.3.6-4 and 6.6.3.6-5 fail because procedure/function parameters have not been implemented.

Test 6.9.4-15 fails because *text* has not been defined.

Error Handling

Number of errors correctly detected: 22

Number of errors not detected: 23 (6 basic causes)

Details of errors not detected: The errors not detected fall into a number of categories:

Tests 6.4.3.3-5, 6.4.3.3-6, 6.4.3.3-7 and 6.4.3.3-8 indicate that no checking is performed on the tag field of variant records.

An assignment to an empty record is not detected in test 6.4.3.3-12.

Tests 6.4.6-4, 6.4.6-5, 6.4.6-7, 6.4.6-8, 6.5.3.2-1 6.6.6.4-7 and 6.7.2.4-1 indicate that no bounds checking is performed on array subscripts, subranges, set operations or the CHR function.

Tests 6.6.5.2-1, 6.6.5.2-2, 6.6.5.2-6 and 6.6.5.2-7 fail because I/O has not been implemented strictly according to the standard.

Tests 6.6.5.3-3, 6.6.5.3-4, 6.6.5.3-5 and 6.6.5.3-6 fail because dispose always returns a nil pointer in B6700 Pascal and no check is performed on the pointer parameter.

Tests 6.6.5.3-7, 6.6.5.3-8 and 6.6.5.3-9 fail because no checks are inserted to check pointers after they have been assigned a value using the variant form of new.

Implementationdefined

Number of tests run: 15

Number of tests incorrectly handled: 1

Details of implementation-dependence:

Test 6.4.2.2-7 shows maxint to be 549755813887.

Tests 6.4.3.4-2 and 6.4.3.4-4 show that the set bounds are 0 and 47. A set of char is not permitted.

Test 6.6.6.2-11 details some machine characteristics regarding number formats.

Tests 6.7.2.3-2 and 6.7.2.3-3 show that boolean expressions are fully evaluated.

Tests 6.8.2.2-1 and 6.8.2.2-2 show that a variable is selected before the expression is evaluated in an assignment statement.

Tests 6.9.4-5 and 6.9.4-11 show that the default size for an exponent field on output is 2; for a real number it is 15 and the size varies for integers and booleans according to the value being written.

Test 6.10-2 indicates that a rewrite on the standard file output is permissible.

Tests 6.11-1, 6.11-2 and 6.11-3 show that the alternative comment delimiters have been implemented, as have the alternative pointer symbols. No other equivalent symbols have been implemented.

Test 6.6.6.1-1 fails because function parameters have not been implemented, and therefore standard functions cannot be used as procedure/function parameters.

Quality Measurement

Number of tests run = 23

Number of tests incorrectly handled = 0

Results of tests:

Test 5.2.2-1 shows that identifiers are distinguished over their whole length.

Test 6.1.3-3 shows that more than 20 significant characters may appear in an identifier, in fact, the number of characters in a line is allowed.

A warning is produced if a semicolon is detected in a comment (test 6.1.8-4).

Tests 6.2.1-8, 6.2.1-9 and 6.5.1-2 indicate that large lists of declarations may be made in each block.

Tests 6.6.1-7, 6.8.3.9-20 and 6.8.3.10-7 show that procedures, for statements and with statements may each be nested to a depth greater than 15.

An array with an integer indextype is not permitted (test 6.4.3.2-4).

Test 6.4.3.3-9 shows that variant fields of a record occupy the same space, using the declared order.

Test 6.4.3.4-5 (Warshall's algorithm) took 0.816461 secs CPU and 143 bytes on the Burroughs B6700.

Tests 6.6.6.2-6, 6.6.6.2-7, 6.6.6.2-8, 6.6.6.2-9, and 6.6.6.2-10, tested the sqrt, atan, exp, sin/cos and ln functions and all tests were completed successfully, without any significant errors in the values.

Test 6.7.2.2-4 shows that div has been implemented consistently for negative operands, returning trunc. mod returns the remainder of div.

Tests 6.8.3.5-2 shows that case constants do not have to be of the same type as the case-index, if the case-index is a subrange, but the constants must be compatible with the case-index.

Test 6.8.3.5-8 shows that a large case statement (>256 selections) is permissible.

Test 6.8.3.9-18 indicates that range checking is always used in a case statement after a for statement to check the for variable.

Test 6.9.4-10 shows that file buffers are flushed at the end of a block but test 6.9.3-14 indicates that recursive I/O using the same file may produce unexpected results.

Extensions

Number of tests run = 1

Test 6.8.3.5-14 shows that the *otherwise* clause in a *case* statement has been implemented according to the accepted convention.

B6700 Pascal - Future Plans and Commentary on Results

The Validation Suite has shown up a number of flaws in the Tasmania B6700 compiler, as documented in the preceding report. We expect that other compilers will typically fare worse in the number of different flaws detected because we have had the benefit of experience (and fixing bugs) as we were developing the suite. This brief document outlines what we expect to do about them.

(1) Minor Flaws

Some of the reported flaws are easy to fix, and have survived to be reported to you only by an oversight, or because the relevant test has only recently been added to the suite. Examples are semicolons in empty records, and incorrect *var* parameter typing. These will be fixed as soon as possible, and probably before this document is released.

(2) Substantial flaws

Two major flaws have survived because they require a reasonable amount of work to repair. These are the deviations of the i/o system, which seem to indicate a revision of the i/o run-time system, and procedure and function parameters which could not have been implemented until the draft standard solution was published due to the insecurities in the original version. These are under revision, and will be fixed shortly. Procedure and function parameters particularly should not take long.

(3) Long-term and medium-term improvements

In the long term, we plan to implement techniques which we have evolved or borrowed for improving the security of Pascal in our compiler, such as checking bounds efficiently outside the B6700 hardware checks, providing correct scoping checks, checking the validity of goto-statements, etc.

R.A.Freak A.H.J.Sale

August 1979

R.A. Freak & A.H.J. Sale

PDP-11 OMSI 1

VALIDATION REPORT

MACHINE: *DEC PDP-11 running RSTS V06C-03*
COMPILER: *OMSI Pascal-1 (Field Test Version X1.2)*
DATE: *1979 September 9 & 10*
TESTS BY: *Barry Smith, Oregon Software*
(Implementation/maintenance team)
ANNOTATED BY: *A.H.J.Sale (1979 September 13)*

CONFORMANCE TESTS

Number of tests attempted: 137
Number of tests passed: 122
Number of tests failed: 15 (13 causes)
Invalid tests discovered: 2

*6.1.8-3 Comment delimiters are not required to be pairwise matching; this makes {This part of the scanner looks for a *) delimiter} a disallowed comment.*

6.2.2-3 Pointer scope is not handled correctly, so that correct programs fail to compile.

6.2.2-8 Assignment to function-identifier from within nested procedure or function generates bad code.

6.4.3.3-1 Empty record types with semicolons and empty case variants are not permitted.

6.4.3.5-2 and -3 An unknown interaction between RSTS I/O on temporary files and the implementation of the run-time support.

6.4.5-9 Equal compatible sets of different basetypes do not compare equal. (Pascal-1 scales the basetype to force a representation of bit 0 on the lowerbound, giving errors in comparisons as shifts are not inserted to compensate. Also set of char is implemented as set of ' '..'←' (64 chars).)

6.6.3.1-5 and 6.6.3.4-2 Only J&W procedural parameters allowed, not the N462 versions. The second test is relevant to the feature actually implemented, but has not been run with modifications.

6.6.5.2-3 Does not check eof on an empty temporary file.

6.6.5.4-1 Pack and unpack not implemented.
6.8.2.1-1 Empty field specifications not allowed in record declarations.
6.9-1 Eoln and eof not correct: relation between RSTS and implementation causes unknown fault.
6.9.2-3 Conversions on reading real numbers are not identical to the conversions performed by the compiler.
6.9.4-7 Writing boolean values is incorrectly right-justified. (AHJS comments that the new draft may change this or tighten up wording.)

DEVIANCE TESTS

Number of tests attempted: 95
Number of extensions: 2 (as stated by B. Smith)
Number of deviations: 41 (25 causes)

6.1.5-4 Allows real number constants without digits after point.
6.1.7-5 and 6.9.4-12 Packed is ignored so that packed array of char is identical to array of char, and similarly with other structures.
6.1.7-6 and -7 The requirements to be a string-type are not checked, allowing deviant programs to execute.
6.1.7-8 The requirements to be a string-type are not checked, together with an obvious error, allows erroneous values to be given to a type.
6.1.7-11 Allows empty string: ie '' is equivalent to packed array[1..0] of char.
6.2.2-4 Incorect scope allows incorrect program to compile.
6.2.2-7 Invalid program executes with (a) function whose identifier is inaccessible and (b) another function has an attempted assignment outside its block.
6.2.2-9 A function-identifier may be assigned to outside its block.
6.3-2, -3, -4, -5, 6.7.2.2-9 Signed characters, strings and enumerated types are allowed.
6.4.3.1-1 Allows packed scalars, subranges, ie not restricted to structures.
6.4.3.1-2 Allows "packed" type-identifier.
6.4.3.2-5 String types are allowed to have non-integer subrange indextypes.
6.4.3.4-3 Set of real erroneously not detected.
6.4.5-2 Var parameters which are compatible but not identical are allowed.
6.4.5-3 and -13 Non-identical array types allowed as var parameters.
6.4.5-5 Non-identical pointer types allowed as var parameters.
6.4.6-10, -11 and -12 Compiles file assignment as descriptor copy, and similarly for records containing file components.
6.6.2-5 Allows function definitions without any assignment to function-identifier.
6.8.2.4-2, -3 and -4 Allows goto statements to transfer into structured statement components.
6.8.3.9-2, -3, -4 and -16 Any assignment to a for-control-variable is allowed inside the controlled statement, and it in fact changes the value.
6.8.3.9-9, -13 and -14 Allows a for-control-variable to be program-global, non-local, or a var parameter.
6.8.3.9-19 Two loops using same variable interact to produce infinite loop construction, and other insecurities.
6.10-1 Ignore program parameters, allowing use of external file not stated.
6.10-3 The files input and output are not implicitly declared at the program level, but at a lexically enclosing level.
6.10-4 The entire program heading, including the reserved word program, may be omitted.

Claimed Extensions:

6.7.2.3-4 And, or and not are overloaded to be a representation-dependent set of operators on integer type.
6.9.4-9 Negative field widths in writes of integers produces octal interpretation in field of abs(width).

ERRORHANDLING TESTS

Number of tests attempted: 48
Number of errors detected: 11 (9 causes)
Number of tests failed: 2

Tests failed:

6.4.3.3-12 Crash at run-time due to empty record-field.
6.6.5.2-2 Relation between RSTS I/O and implementation run-time support.

Errors detected:

6.4.6-6, 6.5.3.2-1 Assignment compatibility: indextype vs subscript value.
6.6.5.2-1 Put not allowed if eof false.
6.6.6.2-4 ln(0.0) or ln(negative)
6.6.6.2-5 sqrt(negative), but continues execution!!
6.6.6.2-2 trunc(largereal)
6.6.6.2-3 round(largereal)

6.7.2.2-3, -8 Div and mod by 0, but continues execution!!
6.9.2-4 Read of textfile, but chars do not represent integer value.
6.9.2-5 Read of textfile, but chars do not represent real value.

Errors not detected:
Use of undefined values.
Variant undefinition.
All assignment compatibility except indextype in arrays.
Nil or undefined pointer dereferencing.
Undefined function result.
File buffer aliasing and use of file.
Dispose of nil or undefined pointer value.
Dispose of variable currently var parameter or with aliased.
Dynamic variant record used in expression or assignment.
Succ or pred of limiting value in type.
Chr of very large integer.
Overflow of integer type.
Assignment compatibility with overlapping sets.
Case expression with no matching label (falls through).
Use of for-control-variable after loop termination.
Nested loops using same control-variable.

IMPLEMENTATION-DEFINED TESTS
Maxint = 32767.
Set of char not implemented, but taken as equivalent to set of ' '..'←'.
Set limits are 0..63.
Standard functions not allowed as functional parameters.
Real representation has 24-bit mantissa, rounds on arithmetic,
eps=5.96e-8; xmin=2.93e-39; xmax=1.70e+38.
Full evaluation of boolean expressions.
Selection then evaluation in a[i] := exp.
Evaluation then dereferencing in p↑ := exp.
Writes two exponent digits in real numbers.
Default field widths integer 7
boolean 5
real 13
Rewrite permitted on output.
Both comment delimiters allowed, no others

Ran 16 implementation-defined tests successfully.

QUALITY TESTS
Number of tests attempted: 24
Number of tests failed: 3

Tests failed:
6.2.1-9 Compiler loops when presented with program with 50 labels.
6.6.6.2-9 Compiler refuses to compile a real expression in sin/cos test due to "lack of registers".
6.8.3.9-20 Compiler crashes after compiling 11 nested for-loops.

Quality measurements:
5.2.2-1, 6.1.3-3 Any length identifiers allowed; disallows all mis-spellings.
6.1.8-4 Unclosed comments swallow text without trace.
6.2.1-8 Allowable number of types ≥ 50.
6.4.3.2-4 Array[integer] diagnosed but message not good.
6.4.3.3-9 Record fields allocated representation space in declaration order.
6.4.3.4-5 Warshall's algorithm timing/space test not yet run.
6.5.1-2 Allowable number of variable declarations ≥ 100.
6.6.1-7 Allowable number of nested procedures must be ≤ 10.
6.6.6.2-6, -7, -8 and -9 Quality tests on sqrt, arctan, exp and ln carried out. Some minor inconsistencies.
6.7.2.2-4 Mod inconsistently implemented for negative operands.
6.8.3.5-2 No warnings for impossible case clauses.
6.8.3.5-8 Allowable number of case-constants ≥ 256.
6.5.3.9-18 Undefined (out-of-range) values of case expressions are possible and are undetected but do no violent damage.
6.8.3.10-7 Allowable number of nested with-statements must be ≤ 3.
6.9.4-10 Textfile without eol at end is still printed.
6.9.4-11 Recursive I/O allowed on same file, and still works.

- -

Pascal P4 on B6700 at Tas

PASCAL VALIDATION SUITE REPORT

Pascal Processor Identification:

Computer: Burroughs B6700

Processor: Pascal-P4 compiler received from Sydney, (March 1979) updated as described below.

The Pascal-P4 compiler was compiled with the B6700 Pascal version 2.9.001 (university of Tasmania) compiler.

The Pascal-P4 interpreter was compiled with the B6700 Pascal version 2.9.178.008 (UCSD) compiler.

Test Conditions:

Tester: C.D. Keen

Date: August 1979

Validation Suite: Version 2.2

Details of Update to Compiler

The original P4 compiler was updated to the level described in the Pascal Newsletter, 13 (Dec 1978).

The compiler and interpreter were both reformatted into 72 character lines.

The declaration of the file 'PRR' was included in the compiler.

The maximum length of constant strings, 'STRGLGTH' was increased to 60 in both the compiler and interpreter.

The compiler was extended to accept both upper and lower case identifiers, with conversion to all upper case characters in 'INSYMBOL'.

The compiler was extended to recognise the alternative form of comment delimiters: '{ ... }'.

The sizes of the integer, real and set constant tables, and the bounds table in the interpreter were extended to 100 words each.

During the processing of the validation suite the following errors in the compiler were corrected:

a) In the procedure 'GEN2T' the field width of bounds variables was increased.

```
Replace:    WRITELN(PRR,FP1:3+ORD(ABS(FP1)>99)*5,FP2:8);
by:         WRITELN(PRR,FP1:14,FP2:14);
```

b) The assignment 'FSP := LSP' near the end of the procedure 'TYP' causes an error if 'LSP' is undefined. This can occur if the type commences with 'packed', but does not have the subsequent symbol in 'TYPEDELS'.

```
Replace:    ERROR(10); SKIP(FSYS+TYPEDELS);
by:         ERROR(10); SKIP(FSYS+TYPEDELS); LSP := nil
```

c) Fix number 17, proposed by Jim Miner (PN, Feb 1978, p71) to prevent comparisons of arrays and records within the procedure 'EXPRESSION' can result in 'TYPIND' not being defined. This also prevents comparisons of strings.

```
Replace:    if not STRING(LATTR.TYPTR)
            and (LOP in [LTOP,LEOP,GTOP,GEOP]) then
              ERROR(131);
            ERROR(134);
by:         if not STRING(LATTR.TYPTR) then ERROR(134);
            TYPIND := 'M';

Replace:    ERROR(134);
by:         ERROR(134);  TYPIND := 'M';
```

d) If the bounds of a for statement are not parsed correctly then the name of the label at the commencement of the for statement will not be defined. This causes the generation of a jump to this label to crash at:

```
... GENUJPXJP( .. ,LADDR); ...
```

To ensure that 'LADDR' is defined, even when syntax errors occur in 'FORSTATEMENT', insert the following statement at the commencement of the body of that procedure:

```
LADDR := INTLABEL + 1;
```

Conformance Tests:

Number of tests passed: 93

Number of tests failed: 46 (22 basic causes)

Details of failed tests:

Test 6.1.2-3 fails because only the first 8 characters of an identifier are significant, and not the identifier's actual length.

Test 6.1.7-2 fails because the maximum length of strings is restricted.

Test 6.2.2-3 fails because the domain of a pointer type ↑T is not permitted to have its defining occurrence anywhere in the type-declaration-part in which it occurs.

Test 6.2.2-8 fails because assignment is not permitted to a function identifier at a lower level than the level at which the function is declared.

Tests 6.4.2.2-2, 6.7.2.2-5 and 6.8.3.9-7 fail because 'maxint' is not predeclared.

Tests 6.4.3.1-3, 6.4.3.5-1, 6.5.1-1 and 6.5.3.4-1 fail because declarations of files are not permitted.

Test 6.4.3.2-3 fails because the character set differed between the compiler and interpreter, so that integer checks on subranges of char are invalid.

Test 6.4.3.3-1 fails because variant parts without tag fields are not permitted.

Test 6.4.3.3-1 fails because a record containing only a semicolon produces a syntax error.

Tests 6.4.3.5-2, 6.6.3.1-3 and 6.9.4-15 could not be completed because 'text' is not predeclared.

Tests 6.4.3.5-3, 6.6.3.1-3, 6.6.5.2-3, 6.6.5.2-4, 6.6.5.2-5, 6.9.1-1, 6.9.2-1, 6.9.2-2, 6.9.2-3, 6.9.3-1, 6.9.4-1, 6.9.4-2, 6.9.4-3, 6.9.4-4, 6.9.4-6, 6.9.4-7 and 6.9.5-1 could not be completed because 'reset' and 'rewrite' are not implemented.

Test 6.4.3.5-4 fails because the output line is not flushed at the program's completion. This is not explicitly performed by the interpreter, but is dependent on the processor used to compile the interpreter.

Tests 6.6.3.1-5, 6.6.3.4-1, 6.6.3.4-2 and 6.6.3.5-1 could not be completed because procedure and function parameters are not permitted.

Test 6.6.5.3-2 could not be completed because 'dispose' is not permitted.

Test 6.6.5.4-1 could not be completed because 'pack' and 'unpack' are not implemented.

Test 6.6.6.3-1 could not be completed because 'round' is not permitted.

Test 6.8.2.1-1 could not be completed because the implementation of scalar types is dependent on the processor which compiled the interpreter, and the UCSD B6700 Pascal compiler does not correctly implement operations on boolean variables. (not is implemented as one's complement)

Test 6.8.2.4-1 fails because nonlocal goto's are not permitted.

Test 6.8.3.5-4 fails because the maximum range of case labels is 1000. (CIXMAX)

Test 6.8.3.9-1 fails because the assignment to a control variable in a for loop precedes the evaluation of the second expression in the for statement.

Test 6.9.4-7 fails because the writing of boolean variables to text files is not implemented.

Test 6.9.6-1 fails because 'page' is not permitted.

Deviance Tests

Number of deviations correctly detected:	52
Number of tests showing true extensions:	4
Number of tests not detecting erroneous deviations:	25 (13 basic causes)
Number of tests failed:	12 (6 basic causes)

Details of extensions:

Tests 6.8.3.9-9, 6.8.3.9-14 and 6.8.3.9-19 show that a for control variable can be globally declared.

Test 6.10-1 shows that file parameters in the program heading are ignored.

Details of deviations not detected:

Test 6.1.2-1 shows that 'nil' is a predeclared identifier, rather than a reserved word.

Tests 6.1.7-6 and 6.4.3.2-5 show that the index bounds of a string are not restricted to 1..n.

Tests 6.1.7-7 and 6.1.7-8 show that strings are permitted to be an array of a subrange of char.

Tests 6.2.2-4, 6.3-6 and 6.4.1-3 show that common scope errors are not detected.

Tests 6.3-5 and 6.7.2.2-9 show that a signed string is permitted as a factor.

Tests 6.4.5-2, 6.4.5-3, 6.4.5-4, 6.4.5-5 and 6.4.5-13 show that type compatibility is used with var parameters, rather than enforcing identical types.

Test 6.6.2-5 shows that no check is made to ensure that an assignment to a function identifier exists in the code of that function.

Test 6.6.6.4-6 shows that 'succ' and 'pred' can be applied to real arguments.

Tests 6.8.2.4-2 and 6.8.2.4-3 show that a goto is permitted between branches of an if or case statement.

Tests 6.8.3.9-2, 6.8.3.9-3 and 6.8.3.9-4 show that an assignment to a for control variable is permitted within a loop.

Test 6.9.4-9 shows that an output field width can be negative.

Test 6.10-3 shows that 'output' can be redefined at the program level, and yet still exist as a file.

Test 6.10-4 shows that a program heading is not required.

Details of failed tests:

Tests 6.1.7-5 and 6.9.4-12 fail because the reserved word packed is ignored.

Tests 6.4.6-11 and 6.4.6-12 fail because declarations of files are not permitted.

Test 6.8.3.9-16 fails because 'reset' and 'rewrite' are not implemented.

Test 6.6.1-6 fails because no check is made to ensure that forward declared procedures and functions are actually present.

Tests 6.6.3.5-2, 6.6.3.6-2, 6.6.3.6-3, 6.6.3.6-4 and 6.6.3.6-5 fail because procedure and function parameters are not permitted.

Test 6.6.6.3-4 fails because 'round' is not permitted.

Error Handling

Number of errors correctly detected:	13
Number of errors not detected:	33 (16 basic causes)

Details of errors not detected:

Test 6.2.1-7 shows that variables are not preset to 'undefined' at interpretation time.

Tests 6.4.3.3-5 and 6.4.3.3-6 indicate that no runtime checks are performed on the tag field of variant records.

Tests 6.4.3.3-7 and 6.4.3.3-8 could not be completed because variant records without tag fields are not permitted.

Tests 6.4.6-7 and 6.4.6-8 show that the determination of compatibility between set types is incorrect.

Test 6.6.2-6 shows that the use of a function without assignment to the function-value-variable is permitted.

Tests 6.6.5.2-1, 6.6.5.2-2, 6.6.5.2-6, 6.9.2-4 and 6.9.2-5 could not be completed because 'reset' and 'rewrite' are not implemented.

Test 6.6.5.2-7 could not be completed because declarations of files are not permitted.

Tests 6.6.5.3-3, 6.6.5.3-4, 6.6.5.3-5 and 6.6.5.3-6 could not be completed because 'dispose' is not permitted.

Tests 6.6.5.3-7, 6.6.5.3-8 and 6.6.5.3-9 fail because no checks are made on pointer variables after they have been assigned a value using the variant of 'new'.

Tests 6.6.6.2-4, 6.6.6.2-5 and 6.7.2.2-3 show that no explicit checks are made by the interpreter for invalid arguments to 'alog', 'sqrt', '/', div and mod.

Test 6.6.6.3-2 shows that no check is made to ensure that the result of a call to 'trunc' is in the range -maxint..maxint.

Test 6.6.6.3-3 could not be completed because 'round' is not permitted.

Tests 6.7.2.2-6 and 6.7.2.2-7 could not be completed because 'maxint' is not predeclared.

Test 6.7.2.4-1 fails because operations on overlapping sets are not detected.

Tests 6.8.3.9-5 and 6.8.3.9-6 show that the use of a for control variable immediately after the loop termination is not prevented.

Test 6.8.3.9-17 shows that two nested for statements with the same control variable are permitted.

Implementation defined

Number of tests run: 15

Number of tests incorrectly handled: 9 (5 basic causes)

Details of implementation-dependence:

Only implementation details fixed in the compiler are considered, and not details dependent on the processor which compiled the interpreter.

Test 6.4.2.2-7 shows that 'maxint' is not predeclared.

Tests 6.4.3.4-2 and 6.4.3.4-4 show that the maximum permitted range of set values is 0..47. This is the minimum required to parse the P4 compiler.

Test 6.6.6.1-1 could not be completed because procedure and function parameters are not permitted.

Tests 6.7.2.3-2 and 6.7.2.3-3 show that boolean expressions are fully evaluated.

Tests 6.8.2.2-1 and 6.8.2.2-2 show that a variable is selected before the expression is evaluated in an assignment statement.

Tests 6.9.4-5, 6.9.4-11 and 6.10-2 could not be completed because 'reset' and 'rewrite' are not implemented. The default field width specifications are: 10 for integers, 20 for reals (floating point notation only) and 1 for characters.

Tests 6.11-1, 6.11-2 and 6.11-3 show that no alternative symbols have been implemented.

Quality Measurement

Number of tests run: 23

Number of tests incorrectly handled: 11

Results of tests:

Test 5.2.2-1 shows that only the first 8 characters of an identifier are significant.

Test 6.1.8-4 shows that no warning is given of semicolons within comments.

Tests 6.2.1-8, 6.2.1-9 and 6.5.1-2 indicate that long lists of declarations can be made in each block, depending on the maximum extent of the heap in the compiler.

Test 6.4.3.2-4 shows that an array with 'integer' index type is not permitted.

Test 6.4.3.3-9 shows that variant fields of a record occupy the same space, using 'reverse correlation'.

Test 6.4.3.4-5 (Warshall's algorithm) took 172.530 seconds CPU interpretation time. (cf. 10.498 seconds CPU execution with UCSD B6700 compiler)

Test 6.6.1-7 shows that the maximum depth of procedure nesting is 10. (MAXLEVEL)

Tests 6.6.6.2-6, 6.6.6.2-7, 6.6.6.2-8, 6.6.6.2-9, 6.6.6.2-10 and 6.7.2.2-4 are only relevant to the implementation of 'sqrt','atan','exp','sin','cos', 'ln', div and mod in the processor which compiled the interpreter.

Test 6.8.3.5-2 shows that case constants must be compatible with the case index, but do not have to be of the same type if the case index is a subrange.

Test 6.8.3.5-8 shows that a large case statement is permitted.

Test 6.8.3.9-18 shows that a range check is performed in a case statement after a for statement to check the value of the for control variable.

Test 6.8.3.9-20 and 6.8.3.10-7 show that for and with statements can be nested to a depth exceeding 15.

Test 6.9.4-10 shows that the flushing of the output line buffer depends on the processor which compiled the interpreter.

Test 6.9.4-14 shows that recursive I/O is permitted, using the same file.

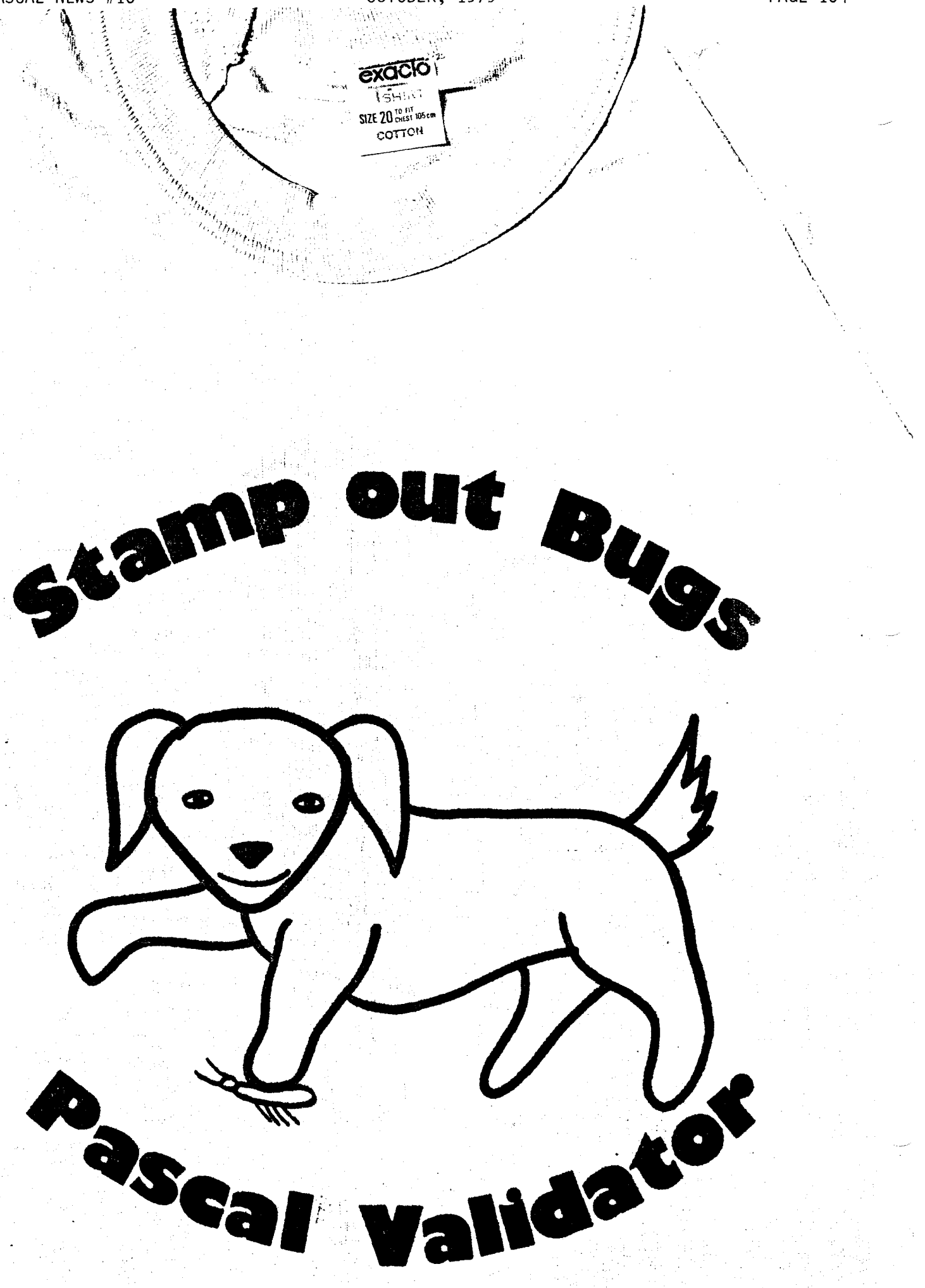
Extensions

Number of tests run: 1

Details of test:

Test 6.8.3.5-14 shows that no extensions have been made to the standard syntax of the case statement.

(* Hurry and send a Validation Report on the Pascal Compiler you use to Pascal News! *)



POLICY: PASCAL USER'S GROUP (79/09/01)

Purposes: Pascal User's Group (PUG) tries to promote the use of the programming language Pascal as well as the ideas behind Pascal through the vehicle of Pascal News. PUG is intentionally designed to be non-political, and as such, it is not an "entity" which can take stands on issues or support causes or other efforts however well-intentioned. Informality is our guiding principle; there are no officers or meetings of PUG.

The increasing availability of Pascal makes it a viable alternative for software production and justifies its further use. We all strive to make using Pascal a respectable activity.

Membership: Anyone can join PUG: particularly the Pascal user, teacher, maintainer, implementor, distributor, or just plain fan. Memberships from libraries are also encouraged.

See the ALL-PURPOSE COUPON for details.

FACTS ABOUT Pascal, THE PROGRAMMING LANGUAGE:

Pascal is a small, practical, and general purpose (but not all-purpose) programming language possessing algorithmic and data structures to aid systematic programming. Pascal was intended to be easy to learn and read by humans, and efficient to translate by computers.

Pascal has met these design goals and is being used quite widely and successfully for:

* teaching programming concepts
* developing reliable "production" software
* implementing software efficiently on today's machines
* writing portable software

Pascal is a leading language in computer science today and is being used increasingly in the world's computing industry to save energy and resources and increase productivity.

Pascal implementations exist for more than 62 different computer systems, and the number increases every month. The Implementation Notes section of Pascal News describes how to obtain them.

The standard reference and tutorial manual for Pascal is:

Pascal - User Manual and Report (Second, study edition)

by Kathleen Jensen and Niklaus Wirth

Springer-Verlag Publishers: New York, Heidelberg, Berlin

1978 (corrected printing), 167 pages, paperback, $7.90.

Introductory textbooks about Pascal are described in the Here and There Books section of Pascal News.

The programming language Pascal was named after the mathematician and religious fanatic Blaise Pascal (1623-1662). Pascal is not an acronym.

Pascal User's Group is each individual member's group. We currently have more than 3357 active members in more than 41 countries. This year Pascal News is averaging more than 120 pages per issue.