Return to:
Pascal News
2903 Huntington Rd. • Cleveland, Ohio 44120
Return postage guaranteed Address Correction requested

Attn: Pascal Group [83]
Univ. of Minnesota
Room 217 Browse Copy
UCC: 227 EX
Minneapolis,MN 55455

Rm 217 Browse Copy $10.00

PASCAL USERS GROUP

Pascal News

Communications about the Programming Language Pascal by Pascalers

- Pascal Processor Validation Procedure
- A Better Referencer
- Use of Generic Capsules
- Implementation Reports
- Validation Suite Reports
- Announcements

Number
25
APRIL 83

POLICY: PASCAL NEWS (Jan. 83)

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

Purpose: The Pascal User's Group (PUG) promotes 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 takes 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 COUPON for details.

- *Pascal News* is produced 4 times during a year; January, April, July October.
- ALL THE NEWS THAT'S FIT, WE PRINT. Please send material (brevity is a virtue) for *Pascal News* single-spaced and camera-ready (use dark ribbon and 15.5 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 — explains 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.

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.

VALIDATION SUITE REPORTS — reports performance of various compilers against standard Pascal ISO 7185.

Pascal News

Communications about the Programming Language Pascal by Pascalers

APRIL 1983 **Number 25**

2 EDITORS NOTES

3 PASCAL USERS GROUP (UK)

3 I.T. and M.I.S.S. *By Phillip Darrington*
4 Pascal-An Effective Language Standard *By Brian Wichmann*
6 Pascal Processor Validation Procedure *By David Blyth*

SOFTWARE TOOLS

12 A Better Referencer *By J. Yavner*
16 The Use of Generic Capsules with the University of Minnesota Pascal 6000 Compiler *By Frank L. Friedman, Alessio Giacomucci, Carol A. Ginsburg and Anita Girton*

ANNOUNCEMENTS

24 PACS Computer Game Festival
24 Oh! Pascal!
24 New Modula-2 Version
25 New Ticomm Microcomputers
25 Edison on IBM Personal Computer
25 JRT Pascal
27 Pascal Compiler for IBM Mainframe
28 Great Plains Announcement
28 INMOS Announces OCCAM
30 Tiny Pascal Plus
30 Help Wanted
30 Ridge Thirtytwo Graphics

32 VALIDATION SUITE COUPON

33 IMPLEMENTATION REPORTS

45 Machine Index

47 VALIDATION SUITE REPORTS

47 HP 3000 Series 33
51 Intel 8085, Zilog 80 (Cogitronics)
52 IBM 370 (AAEC)
56 Pascal 1100
58 IBM 4341
60 VAX 11-780

62 BACK ISSUE COUPON

64 MEMBERSHIP COUPON

Good Members Hello;

I now have control of most elements of Pascal News and future submission articles, comments and good jokes should be addressed:

Pascal News
2903 Huntington Road
Cleveland, Ohio 44120

Our United Kingdom and European elements are thriving and boisterous.

PUG (UK)	PUG (Eur)
P.O. Box 52	ARGE Pascal
Pinner	Hellmut Weber
Middlesex HA5 3FE	Degenfeldstrasse 2
U.K.	D — 8000 München 40

These groups should be excellent sources of local and international information.

We have lost an element and have no successor. PUG (Aus) has experienced increased costs and decided PUG (USA) could support them with little loss of timeliness. I would like to thank them for their past performance. I am sorry I did not have the opportunity to work with them.

PUG (USA) will now serve various needs.

We now serve inside USA and outside USA members and also provide an air mail option for those who need Pascal News as quick as possible.

Writing of timeliness I am reminded that the newsletter has deadlines. These are January 1st, April 1st, July 1st and October 1st. When you have material for the newsletter please send it as quick as possible. Do not worry about the deadlines but keep in mind news loses its value as it matures.

I will continue to publish implementation notes and announcements of the trade. I encourage members and vendors to test drive their new compilers with the "Validation Suite". Send the reports to me and then we will all know the best performing compilers.

I have been asked if we would pay for articles. I have thought about this and worried where I would get the money.

I have decided to accept advertising and use this money to pay honorariums to writers of good articles.

A reminder that back issues will reflect higher reprinting costs and have a $25 per set price after July 1st

Its still a bargain at $15 now.

One more thing. Thank you for your renewals and lovely comments. I have been encouraged by your thoughts.

Charlie

Dear Pascalers,

here we are reopening PUG Europe:
Lor + Martha +
Erwin + Hellmut +
Jurgen + Manfred +
Urf (Korbinian).

We are Pascal fans and users from the university and industry who are organizing in our spare time the distribution of Pascal News for the European region.

From our viewpoint, being mainly Pascal users, we would like to encourage you to help in keeping Pascal News a living forum, a market place for all Pascal users. So here again is a call for papers and programs. There are certainly many tools, especially for textprocessing, which are of interest for the Pascal community, maybe for simple use, maybe in order to compare ideas about problems which many of us may already have encountered. And think about all the programs for solving the daily commercial problems.

Another subject which we think important is documentation. If you have to (or like to) use non-standard features mentioning those increases portability. An extreme example of the necessity of documentation are two Pascal implementations which use one name (append) for two non-standard predefined procedures doing different things (append one string to another versus open a textfile for appending text).

To increase the market place function of Pascal News we should like to ask everybody who provides a Pascal source for publication to state whether he/she is willing and/or able to distribute this source in machine readable form (or even as a well readable listing) and if so at what cost.

Lastly we would like to ask all those wishing to contact us to use our official address:

ARGE Pascal
Hellmut Weber
Degenfeldstrasse 2
D — 8000 München 40

and *not* to send registered letters. (We had some problems, as there is no Mr. Pascal to claim them.) If you want to send us money for subscription please use our postgiro account Munchen 51589-801 or send an Eurocheque and please take note that any other form of payment means additional paperwork for us.

Stay happy with Pascal!

Pascal Users Group (U.K.)

Pascal News 23a is a supplement, to plug the lengthening gap between US originating 23 and 24. Readers will note that its contents are quite different from those of previous editions. There is a shift of emphasis from matters of concern at leading edge University level, to those of concern to producers and users of inexpensive standardized products.

That shift has been wholly dictated by the content of material submitted for publication. Whether it is a temporary side-step or a permanent change, will also be decided by contributors (to future editions). PUG(UK) is the servant of you the subscribers and as such, will publish material originating from any section of the user community.

We are all indebted to each contributor but Tony Heyes's generosity in offering his Bibliography suite of programs for refinement through the medium of PN is particularly appreciated. Constructive critiques are welcome.

There is a widening of the user base and an overdue deployment of resources to that end, evidenced by the complementary nature of articles from widely differing sources. Read on and judge for yourselves. Although you will find that 23a is pitched at quite a different level from that of your usual expectations of PN, I sincerely hope that you will welcome it as a stop-gap until 24 becomes available from Rick, Andy, and Co.

The following is offered as an illustration of the scene which prompted the production of a supplement.

Intrigued by advertising which referred to "mere humans", I went along to the personal computer show at the Barbican on September 12th.

Imagine the disappointment at failing to find anything innovative or even mildly interesting. Discovered that with a single exception, exhibitors did not know whether standard Pascal was implemented on the machines offered to the public. More than one of those asked, replied "Yes, it's called UDCS or something like that". At one stand, sponsored by British Petroleum, the Department of Trade and Industry, the Council for Educational Technology, and others, an 'expert' merely looked blank and suggested that I ask someone else. 'Someone Else' replied "We are only interested in things for use in Education". At the National Computing Centre stand, another expert, when asked if his stand offered any information about standard Pascal and its implementation or use in a microcomputer environment, replied "No, there is no demand", deftly followed by "Can I help you sir?" to someone standing behind me. In some instances, the initial answer was "Yes", followed by misrepresentative flannel when a demonstration was requested.

Met a guy who holds a powerful position in the largest education authority in Britain. He believes that BASIC is an "appropriate" language for the "mass" of young people who "won't bother" to become seriously interested in the technology. I should admit at this point, that had my first experience of a perception of machine intelligence been through the medium of BASIC (or COBOL, FORTRAN, etc.), I might easily have joined the ranks of those who either "won't bother" or are suitably unimpressed by obscure combinations of hunches, guesses, and a dash of perceptual skill which only occasionally fails.

PUG

I.T. and M.I.S.S.

From Wireless world.
Reproduced with Phillip Darrington's permission

One of the aims of Information Technology Year and the Microelectronics Education Programme is to involve schoolchildren in the use of microcomputers and related electronic devices. There are the M.E.P., the Micros in Schools Scheme, exhibitions and events throughout the year and beyond. It is, perhaps, fortunate that Mr. Callaghan happened to be watching television on the evening the programme "Now the Chips are Down" was broadcast and was spurred into action then, or we would probably find the propaganda even more frenetic than that now being put out by the energetic Mr. Baker, the prophet of IT.

Information Technology is a curiously diffuse name for a Year. The official definition, "the acquisition, processing, storage, dissemination and use of vocal, pictorial, textual and numerical information by a microelectronics-based combination of computing and telecommunications" appears to encompass most of the activities of the average person, except eating and one or two other processes, although the use of a computer is not often considered essential to the more basic of these.

So far as its involvement of schoolchildren is concerned, the publicity is decidedly shrill, the Minister's aim being to have a computer in every secondary school by the end of the year and even to think about providing them for primary schools.

There can be no argument that young people must be aware of computers and how to use them, but it does seem possible that the present blaze of publicity tends

to obscure the point that computers are a means, not an end. There is also the question of how the micros are to be used in schools.

According to the fifth edition of the Concise Oxford Dictionary (now, admittedly, modified), a computer is "a calculator — an electronic calculating machine" — an unfortunate description, taken too literally by at least some of those responsible for introducing youngsters to computing, with the result that the school micro is often given to the senior math teacher to guard with his life, presumably on the grounds that computers are electronically mathematical and possess no relevance to any other subject.

In other schools, the computer is treated as a kind of totem, and the pupils are taught "Computer Studies". As a subject, computing (meaning programming) is a singularly empty one, unless the pupil learning it intends to become a programmer. A computer is an aid to the process in which it is used — in this instance, learning — and an element of transparency to the user rather than an obscuring of the subject by undue attention to the computer must be the aim.

Clearly, an overnight transformation, after which every teacher would be using a micro as to the manner born, is hardly feasible. But, until the school micro (or one of its terminals or even a micro owned by a pupil or teacher) can be used naturally, as is a dictionary or pocket calculator or a video recorder, it will dominate the learning process. Utmost priority should be given to teachers from all disciplines, from home economics to athletics, to use the computer as an aid, rather than as a distraction, so that pupils who are not to specialize in science or engineering can see that it is of advantage to them to be at ease with computers, but no more than that.

The Inner London Education Authority is aware of these problems and is educating teachers in the use of computers so that, even though there may be only one micro or terminal in the classroom, the pupils will learn the place of a computer by, to use ILEA's word, "osmosis". However, there is evidence aplenty that education authorities in other areas are either hypnotized or revolted by the new equipment and, accordingly, either enshrine it or pass it to the school computer fanatic to impress people with.

In short, a computer is a useful tool, but that is all it is: it can help or it can dangerously hinder learning, and only the education of teachers in its natural use as an aid can decide which. **PUG**

Pascal — An Effective Language Standard

Brian A. Wichmann,6/5/82

Over the last few years, the programming language Pascal has grown in popularity very greatly. It is widely used for teaching in Universities, is available on most micro-processors and main-frames as well. In fact, Pascal is one of the few languages that form a bridge between microprocessor systems and the main-frame world.

Until recently, there has been one drawback to Pascal as a general purpose software tool. The definition of the language was not very precise and in consequence, the portability of Pascal programs was problematic. The British Standards Institution (BSI set up a group under Dr. Tony Addyman to produce a standard definition of the language. This was later superseded by an ISO group also under Tony Addyman. Last October, ISO agreed to the standardization of Pascal, and after editorial work on the document, BSI published the Standard in February of this year (BS 6192).

What does this mean for users of Pascal? The portability of Pascal programs should be much improved provided suppliers implement the Standard and users write their programs to conform to the Standard. One might think that the position with Pascal is no different from that of COBOL or FORTRAN and yet portability problems arise with these languages. There are several reasons for believing that Pascal is different:

1. The Pascal standard is more comprehensive than that of COBOL or FORTRAN. For instance, the COBOL and FORTRAN standards do not require that an invalid program is rejected by a compiler. The Standard for these languages is just a definition of a language rather than a set of requirements for a compiler. This is clearly not very satisfactory since we all write incorrect programs on occasions.
2. The Pascal Standard is simple and devoid of a multitude of options. If the language has lots of options, then program portability is reduced because a program may not be valid without a specific option. COBOL has a large number of options and FORTRAN 77 has two major levels (essentially distinct languages) whereas Standard Pascal has just one option, affecting only one part of the language. This option is to allow procedures to handle arrays whose size varies from call to call. This option, level 1 Pascal, would allow Pascal programs to call FORTRAN routines in many systems.
3. The Pascal test suite is more searching than that of COBOL and FORTRAN. This is essentially a consequence of the definition of the language. The National Physical Laboratory has been collaborating with the University of Tasmania on the construction of this suite for over two years. About 400 copies of the test suite have been sold worldwide. A new version of this suite has recently been issued to correspond to the new ISO Standard. Unlike the COBOL and FORTRAN test suites, the one for Pascal includes incorrect programs which must be rejected: ones to examine the error-handling capability of a compiler, and the "quality" of an implementation. The quality tests indicate if there is any small limit to the complexity of programs that a system can handle and also assesses the accuracy of real arithmetic.

All the major components to make Pascal a good Standard are now available, that is, a Standard definition and tests to verify conformance of a compiler to the Standard.

A Standard and tests to check conformance to the Standard are not alone quite sufficient. The test procedures must be used and results made known to those using Pascal compilers. This can be achieved by independent testing of compilers which is currently being investigated by BSI (Hemel Hempstead). BSI have a wealth of experience with testing other goods but this is their first venture into computer software. For this reason, both NPL and NCC are assisting BSI in this important development.

The last step in this process is to encourage users to request a *Standard* compiler from the suppliers and for suppliers to meet that demand. As a contribution to this last step, NPL held a conference on this topic with its collaborators. Professor Arthur Sale from the University of Tasmania addressed the conference making it an international event. The other key speakers were John Charter from BSI who described how a validation service run by BSI would work. Professor Jim Welsh from UMIST who described how the Standard can be implemented and Lyndon Morgan from NCC who described a guide written to support the test procedures. Also Barry Byrne, from ICL explained how the provision of a standard compiler for Pascal is advantageous in both marketing and for internal use. Mr. Ken Thompson from the European Commission explained the usefulness of international standards within the Community and some of the problems in their effective exploitation.

This program contains five errors, often undetected by compilers. Can you spot them?

```
program test;
const
   nil = '0';
begin
   if nil # '0' then
      writeln( 'WRONG', +nil, .123)
   else
      writeln( 'RIGHT' )
end.
```

Try it on your system and see how many errors are detected.

```
Errors
1. program must contain output as parameter.
2. nil cannot be used as an identifier (it is a
     reserved word).
3. # is written as <> (not equals).
4. nil cannot follow a sign.
5. a decimal point must follow a digit.
```

The corrected program is:

```
program test(output);
const
   nill = '0';
begin
   if nill <> '0' then
      writeln( 'WRONG', nill, 0.123)
   else
      writeln( 'RIGHT' )
end.
```

Although this test is only an illustration, it does show the wide ranging capabilities of current compilers. The results of compilers tested so far can be summarized thus:

Compiler	Errors detected	Accuracy of error messages	Recovery from last error
A	4	3	4
B	2.5	2	3
C	2	2	2
D	1	2	1
E	2.5	3	2
F	3.5	3	3
G	4.5	4	3
H	5	4	4
I	3.5	1	2

All the marks are out of 5. The half marked for detecting an error indicates that the error message was confusing enough for it to be unclear if the error was properly detected. Naturally, the last two columns are subjective. **PUG**

Article formed the basis of piece in *Computer Weekly* by Phillip Hunter. 11th Feb. 1982 page 14

PASCAL PROCESSOR VALIDATION PROCEDURE

By David Blyth
Standardization Office,
National Computing Centre

1 Introduction

Few Pascal users can be unaware of the recent publication of the British Standard for the language which will shortly be adopted internationally. Many users have heard of the suite of validation programs, developed by the University of Tasmania and the National Physical Laboratory, which can be used to check on the standard-conformance of an implementation. This suite is readily available and any user who has a copy can use it to test his own compiler or interpreter. For those brave users who undertake such testing this article presents a brief guide to the steps involved and draws upon experience gained at NCC in a joint NPL/NCC/BSI project to develop and document the validation procedures.

2 The Pascal Standard and Validation Suite

The Pascal standard defines the language itself and the manner in which Pascal programs are to be handled by an implementation. The validation suite contains over 400 test programs whose purpose is to check whether or not an implementation accepts the language as defined in the standard and whether or not programs which are accepted behave as the standard says they should. The standard and the validation suite have been developed in parallel with the result that the suite will provide an exceptionally strenuous test of any implementation. An implementation which performs well under test can be used with confidence in its conformance and reliability.

The suite contains eight types of test program which investigate respectively, conformance, deviance, implementation-defined features, implementation-dependent features, error handling conformance arrays, quality and extensions. These classes of tests are quite distinct and are used in characteristic ways.

2.1 Conformance Tests

Conformance test programs attempt to check that an implementation provides those features required by the standard and that it does so in the manner which the standard specifies. These programs are all correct standard Pascal. If the implementation conforms to the standard these programs all compile and execute. If a conformance test program fails then it is an indication that the implementation does not conform to the standard.

2.2 Deviance Tests

Deviance test programs check whether

(i) the implementation provides an extension of Pascal;
(ii) the implementation fails to check or limit in an appropriate manner some feature of Pascal;
(iii) the implementation incorporates some common error.

No deviance test program is standard Pascal. Each such program contains exactly one such deviation. When a deviance test is run the results are inspected for evidence that the implementation does in fact detect the deviation. If it does not then the implementation does not conform with the standard.

2.3 Implementation-Defined Features

The standard defines an implementation-defined feature as one which may differ between implementations but which is defined for any particular processor. A conforming implementation must be accompanied by a document that provides a definition of all its implementation-defined features. The test programs for implementation-defined features are intended to show how these features are handled in any particular implementation. If they aren't handled in the manner claimed then the implementation does not conform.

2.4 Implementation-Dependent Features

An implementation-dependent feature may differ between implementations and is not necessarily defined for any particular implementation. Here the implementor can either state in his documentation that use of such features is not reported or else have the implementation issue some diagnostic for which such a use is encountered. The test programs in this area are designed to determine the behaviour of the implementation. The implementation conforms only if it behaves as claimed or reports implementation-dependent usages.

2.5 Error-Handling

An error is defined, in section 3.1 of the standard, to be a violation by a program of the requirements of the standard that the implementation is not obliged to detect. An implementation only fails to conform in respect of error-handling if it fails to process an error in the manner claimed in the documentation. The error-handling tests each present the implementation with one error with the aim of determining exactly what the implementation does with it.

2.6 Conformant Arrays

An implementation may conform with the standard at level-0 or at level-1. In plain terms it can either have conformant arrays or it can't. If conformant arrays are provided then all of the features specified for them must be provided according to the standard.

The conformant array tests are a collection of conformance, deviance, implementation-defined, implementation-dependent, error-handling and quality tests designed to test the conformant array features in isolation.

2.7 Quality

Many aspects of an implementation are beyond the scope of the standard, but it is still useful to investigate them. Quality tests explore these areas and investigate:

(a) The limits on the size and complexity of programs imposed by the implementation
(ii) the amount of store needed to perform certain well-defined tasks
(iii) the accuracy of real arithmetic
(iv) the meaningfulness of diagnostics for common types of error
(v) the speed of the code produced.

Quality tests often throw up some surprising results!

2.8 Extensions

Many implementations offer extensions to the standard. The extension tests see whether common extensions (eg those approved by PUG) are implemented.

Together the test programs provide a very thorough test of an implementation.

3 Using the Validation Suite

3.1 Distribution Format

The validation suite is distributed on 9 track magnetic tape with characteristics as follows:

Recording density : 800 or 1600 bpi
Recording mode : NRZI or PE
Character code : ISO 646 or EBCDIC
1200 bytes/block, 80 characters/record.

A purchaser of the tape can specify which density, recording mode and character code he wants.

There are 49 files on the tape. Three of these contain documentation. The rest contain the validation programs.

3.2 Media Conversion

Users whose machines have tape drives should experience no significant problems in reading the distribution tape. Their only concern will be with lexical conversion if necessary.

Users with floppy disc based systems need to do a media transcription to get the suite in a form in which they can use it. This conversion can be tricky, and is almost always done on an ad hoc basis for the particular system concerned.

3.3 Lexical Conversion

There are two character sets to consider when using the suite — the one used to encode the test programs, and the one used to represent "char-type" values on the target computer.

Roughly speaking any consistent set of lexical substitutions can be made, but some may render specific lexical test programs, and some programs which test the char type, irrelevant in validation.

Care is needed to ensure that lexical conversion is consistent throughout. This is particularly important if media conversion affects character code representations.

3.4 Integrity Checking

Following media and lexical conversion it is advisable to check that no corruption has occurred. For this purpose a program called the Checktext program is supplied. It produces a 96-bit binary check pattern using an algorithm originally developed for use in data transmission (CCITT Rec. V.41)

The Checktext program operates on a standardized internal representation of the program and will not be affected by legal lexical substitutions. Certain parts of the program may need customization for use on particular systems and the source code is marked to show where such changes should be made.

The results of the Checktext program should be compared with standard results contained in the User Guide to the suite (supplied with the distributrion tape) and if there is any discrepancy then transcription has introduced errors.

3.5 Checking Validation Suite Assumptions

A validation suite must necessarily make certain assumptions about the nature of the implementations which it will be used to test. The Pascal validation suite assumes that

- text files
- character-strings
- the real-type
- local files

are all implemented, also that

- lines up to 72 characters long can be accepted
- lines up to 72 characters long may be output
- the value of maxint is > 32,000
- the relative precision for reals is < 0.001
- the characters needed to encode the test programs are all accepted as distinct by the implementation
- the "largest" procedure in the test suite is accepted by the implementation (except for certain quality test procedures).

A further implicit assumption is that the real arithmetic system is susceptible to investigation by certain types of method.

The validation suite contains a program called the "Check Assumptions" program which enables the user to determine whether or not the implementation violated any of the assumptions listed above.

4 Planning and Running the Tests

4.1 Planning is Important

Testing an implementation is not just a matter of running all the test programs. The test suite is large and on some machines it is not possible to run all the tests without breaking the suite into batches. Furthermore close attention must be paid to ensure that the behaviour of the implementation is accurately recorded throughout the test procedure. Finally provision must

be made to make it easy to re-run any particular test after preliminary interpretation of test results.

Choice of the method of working can have a marked effect on the overall time taken to run the tests. There are two areas to consider. First some method must be chosen to extract test programs from the files which contain them. Second the organization of the jobs which run the test programs must be decided. The User Guide illustrates three approaches for each of these methods which will cover most cases on a wide range of machines.

Some programs may prove to be rogues on certain implementations. There is no way of knowing in advance which programs will behave in this way for any given implementation. The user should take care so that such programs do not cause the loss of accumulated test results.

In any event some programs will need re-running because the results on the first run may have been inconclusive. The circumstances in which a re-run is needed are given in the Guide.

5 Reporting Results

It is desirable to adhere to a standard form of presentation when reporting the results of a validation. This offers two main advantages.

First, when a formal validation is being done, a standardized report:

1 Processor Identification
2 Test Conditions
3 Conformance Test Results
4 Deviance Test Results
5 Error-Handling Test Results
6 Implementation Defined Test Results
7 Implementation-Dependent Test Results
8 Level 1 Test Results
9 Quality Test Results
10 Extension Test Results

Guidance on the content and presentation of these sections is included and a sample validation report is included as an Appendix.

6 Practical Use

The present article offers only a brief sketch of the validation procedure. At first sight it may look somewhat daunting. In practice the key is attention to detail. The User Guide gives fairly detailed advice on transcription and test job organization, and will be found helpful by most people undertaking tests of implementations. Once transcription and organization have been sorted out the tests usually run smoothly. Carrying out a full test is a rewarding exercise which offers many lessons to language implementors. It is hoped that users and implementors alike will use the test suite and help to promote rapid practical standardization of Pascal.

PUG

Dear Nick,

After our phone conversation the other week, I was rather more relieved to feel that here in the UK there are other Pascalers at work and that PUGUK is viable again. The gap has been too long, and I wish you well in trying to get it going again. I shall try and do what I can and particularly with public domain software, but at the moment, I don't have a great deal of time to spare, nor any telecomms equipment to plug into my computer.

I enclose a cheque for 9 pounds for subscription. On the question of back numbers, I have copies of 12-16, and any subsequent or previous issues would be very welcome. I would have thought that for 17-21 which you already have, it would be worth while putting a note in the next issue to see how many people want them, and then have your printer print adequate copies in total. Much better than spending your time collating everyones' needs and doing individual photocopies of bits and pieces. Perhaps if other people were able to lend you some of the older copies, the same could be done. I'd certainly lend you 12-16 if you like. After all, its the information that matters, not whether the issue is an original or not unless we have an collectors among us. Anyway, mark me down for any back issues you can get your hands on, please.

I am now using Pro-Pascal from Prospero Software as my major programming tool, as well of course as Wordstar to compose programs and write letters. The hardware is OEM kit from Sirton Computers in Purley, by the name of Midas and is in essence an Integrand 10-slot S100 case with PSU, Ithaca IEEE S100 cards (MPU-80, FDC-2, 64KDR and VIO boards) giving 64k and 4Mhz Z80A with CP/M, plus 2*YE-DATA 174D 1Mb drives. The printer is a Qume (a luxury really), and a Volker-Craig VC404 completes the outfit.

I will try and compose a critique of Pro-Pascal as soon as possible, but version 1.4 is due out soon with 8 byte longreals among other goodies. I have written to Charles Foster of Pascal/Z User Group asking if he or his contributors would permit the distribution of any of their Pascal sources to PUGUK members appropriately modified to BS 6192, or if indeed there is any other Public Pascal around in the States. I think we ought to be prepared to reciprocate on this, don't you?

In converting from programming mainly on mainframes in Fortran and having a nodding acquaintance with Cobol, Basic and other languages, there are times when even Standard Pascal has its limitations. Therefore, I've thought of two ways of improving the language. As PUG may have some influence with the powers that be, I've taken the liberty of including the suggestions — by all means put them in a news-letter if you like. I don't believe in trying to persuade compiler-writers to augment their compilers as their job is to implement the standard. If the language is to grow, and if any such need is identified, then it's the standard that must mature. Now BS 6192 is published, it will be some time before any further thought is applied to the subject I expect, if ever, so perhaps now is the time to see if anyone is interested.

John R Logsdon
18 Darley Road
Manchester M16 ODQ

```
Tongue-in-cheek Pascal Language enhancements.

a)   Structured constants.

     Program make-up to be for example:

     PROGRAM example;
     CONST onehundred=100;
          .................... etc
     TYPE
       scalartype=(coffee,jam,bread,tea,biscuit,suicide);

          extype=RECORD
                   a:integer;
                   b,c:char;
                   d:array[0..3] of integer;
                   f:scalartype;
                   g:set of scalartype;
                   h:array[1..20] of char
                 END;
                      .................... etc
     TABLE exl:extype=
               onehundred,'a',chr(20),(0,25,50,75),jam,
               [coffee,tea,bread],'cholesterol';

     VAR exvar:extype;displayl:char;

     BEGIN
       exvar:=exl;
       displayl:=exl.h[4];

          ........................ etc
```

Note the use of the 'chr' function to set up unprintable characters, the absence of any delimiter other than those already used in Pascal and the access of a constant array element. There is no reason why 'ord' should not also be included so that portability is enhanced. The syntax follows closely on that of Pascal as it is and involves no ambiguity in type declaration implicit where structured constants are declared in the constant section as in some implementations. Pointers declared in the correspnding type declaration may be set to whatever internal value represents nil, however they are named and uncompleted arrays of char initialized to spaces.

Such a feature will provide geniune structured read-only constants without the ugly initiation presently necessary in Pascal. In fact, in practice it is easier to put records for initialization in a parameter file and read them in, which does not seem an elegant solution. For micros with restricted memory, initializing a record from constants needs up to two copies of every element — one dynamic and one in the constant area, which is rather wasteful of space.

```
b)   Type-change function.

     Syntax to be, for example:

     PROGRAM another;

     CONST ...................... etc

     TYPE score=(first,second,third,fourth);
          fruit=(apples,pears,oranges,grapes);

     VAR thisscore:score;thisfruit:fruit;

     BEGIN

          {calculate thisscore somehow}

       thisfruit:=fruit(thisscore);

          ........................ etc
```

This facility will provide a logical completion to the built-in functions 'ord', 'chr' and provide a much more readable alternative to the use of variant records. Although there is no reason why the method should not be available for records if the matching of record lengths were entirely the programmers responsibility, there is an objection in that the internal representation of variables will be machine-dependent. I envisage this type-change function purely for scalar variables between scalars and perhaps for pointers between pointers. It is of course really a mechanism to cause the compiler not to check types.

(This facility is similar to one available in AAEC Pascal 8000 for the IBM 360/370 series, and attributed to Kludgeamus)

If any readers have any comments for or against, perhaps PUG can help to air views?

HELP!

Dear Nick;

Systems Used

(i) Apple (II) UCSD Pascal.
(ii) To be delivered December 1982: Burroughs B21-5 (384 K Byte). Pascal ISO draft 5.

Special Interests

Business systems. Particularly rapid access to unsorted data items. Data base management systems.

Information Please

We would be interested in knowing of a Pascal compiler to interim ISO standard or UCSD for Burroughs B1955 with 0.5M Byte working store. Manufacturer does not support Pascal for.

Mr. P A E Herring
MAPAC
17 Market Square
Leighton Buzzard
Bedfordshire
LU7 7EU

Dear Nick,

CET TELESOFTWARE PROJECT

Thank you for your letter of 6th December.

I think you must have got the wrong impression from my letter of 3rd December. We certainly do not want to see a different telesoftware format for PASCAL. As I understand it, the only problem with the cur-

rent format is the TAB character which lies outside the PRESTEL character set. You may be interested in our recent extensions to the format (copy enclosed) which overcome this.

As far as including PASCAL programs in our library is concerned, all I am saying is that we need to learn how to walk before we can run. We are keen to include programs in languages other than BASIC, including PASCAL, but need to be sure there are people who can receive them on our system and will find them useful, before putting them up.

If you know of PASCAL programs which will run on the micros most used in educations, ie 380Z, Apple, Pet, Acorn and TRS 80, I would be interested in receiving details.

Chris Knowles
Telesoftware Project Manager
Council for Educational Technology
3 Devonshire Street, London W1N 2BA

Dear Pascal User,

Please find enclosed details regarding Version 3.1 of the Pascal Validation Suite which was released on the first of October 1982. Should you wish to receive a copy of the suite, please fill in the enclosed application form for a license and send it together with your remittance to:

Dr. Z. J. Ciechanowicz
Division of Information Technology & Computing
National Physical Laboratory
Teddington
Middlesex TW 11 OLW England

On receipt of the form and remittance we will send a magnetic tape containing the suite.

The cost of the package is $100 sterling (+ 15% VAT for UK users) and cheques should be made payable to "The National Physical Laboratory" quoting our reference number NPS 2/01.

Z. J. Ciechanowicz
Division of Information Technology & Computing
Department of Industry
National Physical Laboratory
Teddington, Middlesex TW11 OLW

PS When requesting the suite please supply the tape format you require:
i.e. 1600/800 b.p.i.
ISO/EBCVDIC code

We generally write our tapes with fixed length blocks, 15 records per block, 80 characters per record.

Dear Nick,

1. Can you recommend a PASCAL for XENIX? (LSI II UNIX)
2. Do you know who distributes the Dutch 'Fres University' version of PASCAL? (in the UK)

Brian Kirk
Robinson Systems
Engineering Limited
Red Lion House, St. Mary's Street,
Painswick, GL6 6QR
Telephone: (0452) 813699
VAT Registration: 302 3124 28

APPLICATION FOR LICENSE TO USE VALIDATION SUITE FOR PASCAL

Name and address of requester (company name if requester is a company)

Name and address to which information should be sent (write 'as above' if the same)

Signature of requester

Date

In making this application, which should be signed by a responsible person in the case of a company, the requester agrees that:

(a) The copyright subsisting in the validation suite is recognized as being the property of the British Standards Institution and A.H.J. Sale;

(b) The requester will not distribute machine-readable copies of the validation suite, modified or unmodified, to any third party without permission, nor make copies available to third parties.

In return, the copyright holders grant full permission to use the programs and documentation contained in the validation suite for the purpose of compiler validation, acceptance tests, benchmarking, preparation of comparative reports, and similar purposes, and the provision of listings of the results of compilation and execution of the programs to third parties in the course of the above ctivities. In such documents, reference shall be made to the original copyright notice and the source.

OFFICE USE ONLY

Signed

On behalf of A.H.J. Sale and the British Standards Institution

---

National Physical Laboratory Teddington Middlesex TW11 OLW Telephone 01-977-3222 Telex 262344

Pascal Compiler Validation Suite

NPL issued version 3.1 of the above suite of test programs on 1 October 1982. These programs permit a user to check the compliance of a Pascal compiler and run-time system with the ISO standard for Pascal (ISO 7185, also BS 6192). The new suite is an extensive revision of version 3.0 and the work has been undertaken in conjunction with Professor A.H.J. Sale of the University of Tasmania. Subsequent revisions to the test suite are likely to be of a minor nature.

The British Standards Institution will shortly be launching a pilot validation service base upon the test suite together with other material.

The test suite consists of about 17,300 lines of Pascal programs plus addition comments on each of the 553 test programs. The programs themselves are divided into a number of classes as follows:

182 programs checking that the features of the Standard are available;
157 programs checking that illegal constructs are rejected by a compiler;
82 programs checking the error-detection capability of a Pascal system;
60 programs checking the quality of an implementation;
40 programs checking for Level 1 Pascal ('conformant arrays')'
16 programs checking the variations permitted by the Standard;
13 programs checking for features defined for each implementation;
3 programs checking for extensions.

B.A. Wichmann
Z.J. Ciechanowicz, extension 3977
For BSI, J. Hatton-Smooker, telephone 0442-3111

A Better Referencer

By J. Yavner
Money Management Systems Inc.

The program which follows was developed from the Currie/Sale procedure cross-referencer published in *Pascal News #17*. Of course, any programmer who looks at someone else's program thinks he could do a better job, but I think that by almost any standard I succeeded, though it took me much longer than Sale's three days. I have an excuse, however: prior to this one, I had never written a Pascal program; my experience with the language comes solely from the articles, standards proposals, and validation suite which have been published in *PN*.

The program is shorter, simpler, and almost certainly faster: It has half as many source lines as the Currie/Sale version, but the format is different, and the number of statements in only 25% smaller. The HP 1000/2015 compiler generated 4604 words of code and static data (the 1000 is not stack-oriented). The procedure descriptor is 25% smaller; the reference descriptor is 97% smaller. The syntax analyzer is more tolerant: missing semicolons do not faze it. The program needs 29.80 seconds and 1376 words of heap to process itself, 356.80 seconds and 5780 words to process the 103-procedure, 4000-line P4 compiler.

The improvement stems from the use of a different data structure: The Currie/Sale referencer is optimized for programs of virtually infinite size, using trees and stack and rings of procedure descriptors and chains of reference descriptors, which allow the procedure database to grow very large with the program's taking all the memory ever manufactured and all the time till doomsday to process it. This referencer, on the other hand, is optimized for small programs, and uses an array of procedure descriptor pointers whose size is fixed by a constant, a quick-and-dirty replacement sort, and sets of reference descriptor flags (two sets, since the program prints the reference data from both viewpoints, caller and callee). As the program to be processed increases in size, memory use increases quadratically, eventually surpassing the Currie/Sale referencer, which started out higher but rises only linearly. Execution time, I imagine, ought to expand similarly. It might be interesting to determine where the cross-over point is.

The program uses the CASE . . . OTHERWISE construct which many processors don't recognize yet. The solution for this problem is to upgrade the processor! An interim fix is to replace the CASE's with IF . . . ELSEIF constructs.

Optional lines

Those lines which begin with the null comment are not vitally necessary to the program and can be removed without seriously affecting its operation. They serve primarily to handle HP1000 extensions.

Lines 19, 21-24, 49-51, 68-71, 522-526, and 551-564 make use of implementation-dependent intrinsics to print processing time and heap usage information. These lines can of course be replaced with the appropriate code to do the job at the target installation or simply left out — like most statistics, they're not really necessary.

Lines 113-116 ignore compiler directives. HP Pascal/1000 has its directives bounded by dollar signs. The format is like strings or comments, and thus is in the spirit of Pascal, but nonetheless the construct must be handled separately.

Line 1 is a compiler directive (another is on line 71). The default output line width is 128, which causes 132-character lines to wrap around even though there are still empty columns on the page.

Lines 307-308 and 345-346 add the HP1000 intrinsics to the pre-defined procedure table. They can be replaced with the appropriate constants for the target installation or removed to make the program conform to the pure standard. The format is as follows: Each procedure name is followed by a space. A hyphen terminates each constant. The last string ends with a procedure name, space, and hyphen, and is then padded with trailing spaces to **ConstLen**. As many strings as necessary can be added at 307 as long as they have corresponding calls at 345.

The directive "external" is recognized by the referencer. Lines 8, 149, 237-238, and 477 could be modified to allow the it to recognize the target installation's directives. The implementation dependency was included primarily to show how this is to be done. The nature of the dependency is such that it can be left in even if the target doesn't recognize it.

Options

This referencer contains a much more efficient **AddIntrinsics** procedure than does the Currie/Sale version (because intrinsics inclusion is not the default for that referencer, while it is for this one). The feature can be disabled by setting **Intrinsics false**. The procedure itself is quite small and can be left in even if inactivated.

The program is designed to print the reference information from the standpoint both of caller and callee. Naturally, twice as much information takes twice the space and twice the time to print. Either table can be disabled separately by means of **CallsTable** or **CallersTable**. Almost all the code for printing the tables is common. As an aside, when both tables are printed it is sometimes difficult to figure out which direction is represented by which table, even though the table's title says "calls" or "callers." One table contains only a single procedure defined at level 0: the main program.

Obviously no procedure can call the main program. Similarly, the other table contains the intrinsic procedures. Obviously they don't make any calls.

The identifiers in the input file are truncated if they are too long to fit into the identifier arrays. The length of these arrays is specified by **IdentLen**. Changing this constant requires corresponding modification to the constants defined on lines 5-8, 83-87, and 210-211.

LineWidth can be set to any appropriate value. Setting it to 80 gives two columns of reference data, which is somewhat hard to read (try setting your terminal width to 2 some time). Setting it to 56 forces the tables to have one reference per line, which is rather vertical but still readable.

MaxProc determines the size of the array of pointers to procedure descriptors, and thus the maximum allowable complexity of the input program and the referencer's static size. If it is set to 64 and the HP-specific intrinsics are removed, there is room for 34 procedures, more than most programs published in *PN* need — more than most programs executable on a processor that can't handle large sets probably need.

StackDepth specifies how many BEGIN/END and CASE/END structured statements imbedded inside the body of a procedure the referencer can handle. Few programmers can create code more complicated than 16 nested structures (the referencer never goes deeper than four), but if desired the stack can be extended easily, since each element in the stack takes only one integer.

Offset is the distance from upper case to lower. The program may be set for EBCDIC by changing this constant to the appropriate value.

One final note: no numbers larger than 32767 are needed by the program. On some processors (such as the HP 1000), significant space can be saved by assigning **MaxInt** to 32767 in the referencer's global constant section.

```
APREF  T=00003 IS ON CR00034 USING 00014 BLKS R=0000

{}$LINESIZE 132$
                                        program Pref(input,output);
label 9999;
const
   BlankIdent    = '                ';
   ProgIdent     = 'program         ';
   FwdIdent      = 'forward         ';
   ExtIdent      = 'external        ';
   Intrinsics    = true { Pre-define intrinsic procedures };
   CallsTable    = true { Print table of references FROM procedures };
   CallersTable  = true { Print table of references TO   procedures };
   IdentLen      =   16 { Significance limit for identifiers. };
   LineWidth     =  132 { Determines number of identifiers per line: }
                        { (LineWidth-IdentLen-22) DIV (IdentLen+2)    };
   MaxProc       =   74 { Maximum number of procedures.  This should }
                        { be set to a convenient set size.            };
   StackDepth    =   16 { Maximum block nesting within a procedure };
   Offset        =   32 { Distance from upper- to lower-case };
{}GeTime         =   11 { RTE return-time-of-day code };
type
{}OneWord        = -32768..32767;
{}InfoRec        = record a,tch,b,initch,c,d,e,f            : OneWord end;
{}TimeRec        = record millisecs,secs,minutes,hours,days: OneWord end;
{}
   Whole         = 0..MaxInt;
   StackRange    = 1..StackDepth;
   ErrorTypes    = (NoProgram,Redefinition,TooManyProcs,Misplaced,
                    TooDeep,LostEnds,LostPeriod);

   IdentRange    = 1..IdentLen;
   IdentSet      = set of IdentRange;
   IdentString   = packed array[IdentRange] of Char;
   IdentPtr      = ^IdentString;

   ProcRange     = 1..MaxProc;
   LowProcRange  = 0..MaxProc;
   ProcSet       = set of ProcRange;
   ProcDesc      = record
      name            : IdentString;
      namecases       : IdentSet;
      level           : LowProcRange;
      score           : LowProcRange { 0 : Out; 1 : In; >1 : Occluded };
      defline         : Whole;
      bodyline        : -1..MaxInt;
      calls, callers  : ProcSet;
      end;

var
{}info              : InfoRec;
{}time              : TimeRec;
{}sec, mil          : Integer;

   line, paren       : Whole;
   alpha,alphadigit  : set of Char;

   ident, prog       : IdentString;
   identcases        : IdentSet;
   heapident         : IdentPtr;
   identype          : (InProcess,Other,Def,Directive,Open,CaseOpen,Close);

   procnum           : ProcRange;
   block             : LowProcRange;
   list              : array[ProcRange] of ^ProcDesc;
   sortlist          : array[ProcRange] of ProcRange;

   bracket           : StackRange;
   stack             : array[StackRange] of Whole;
{}
{}procedure Exec(code:OneWord; var time:TimeRec); external;
{}
{}procedure GetInfo $ALIAS 'ΦGMBI'$ (var info:InfoRec); external;

procedure Read; forward;

procedure ReadIdent;
{ Read next identifier from input, keeping track of parenthesis }
{ and skipping comments, quotations, punctuation marks,          }
{ numbers, and compiler directives.                              }
{G:alpha,alphadigit,BlankIdent,ExtIdent,FwdIdent,*ident,*identcases, }
{ IdentLen,IdentRange,*identype,input,Offset,*paren,!Read         }

   const
      ProcIdent  = 'procedure       ';
      FuncIdent  = 'function        ';
      BeginIdent = 'begin           ';
      CaseIdent  = 'case            ';
      EndIdent   = 'end             ';
   var
      j  : IdentRange;
      ch : Char;

   procedure SkipDigits;
   { Skip numeric characters }
   {G:!Read }

      begin while (input^>='0') AND (input^<='9') do Read end;

   begin { ReadIdent }
      ident      := BlankIdent;
      identcases := [];
      identype   := InProcess;
      repeat
        ch := input^;
        if ch='(' then begin { Parenthesis or comment }
          Read;
          if input^<>'*' then paren := paren+1 else begin { Comment }
            Read;
            ch:='{';
            end;
          end;
        if ch=')' then paren := paren-1;
        if ch='''' then repeat Read until input^='''';
{}      if ch='$' then repeat { Compiler directive }
{}        repeat Read until (input^='$') OR (input^='''');
{}        if input^='''' then repeat Read until input^='''';
{}        until input^='$';
        if ch='{' then repeat { Comment }
          while (input^<>'}') AND (input^<>'*') do Read;
          if input^='*' then Read;
          until (input^='}') OR (input^=')');
        if (ch>='0') AND (ch<='9') then begin { Number }
          SkipDigits;
          if input^='.' then begin { Decimal }
            Read;
            SkipDigits;
            end;
          if (input^='E') OR (input^='e') then begin { Exponent }
            Read;
            Read;
            SkipDigits;
            end;
          end
         else if input^ IN alphadigit then begin { Identifier }
          j := 1;
          repeat
            if ident[1]<>' ' then j := j+1;
            ident[j] := input^;
            Read;
            until NOT (input^ IN alphadigit) OR (j=IdentLen);
          for j := j downto 1 do if ident[j] IN alpha
           then begin { Convert to lower case }
            ident[j]   := chr(ord(ident[j])+Offset);
            identcases := identcases+[j];
            end;
          identype := Other;
          if input^ IN alphadigit
           then repeat Read until NOT (input^ IN alphadigit)
           else if (ident=ProcIdent)OR(ident=FuncIdent) then identype:=Def
           else if (ident=FwdIdent) OR (ident=ExtIdent)
            then identype:=Directive
           else if ident=BeginIdent then identype:=Open
           else if ident=CaseIdent then identype:=CaseOpen
           else if ident=EndIdent then identype:=Close;
          end
         else if ch<>'(' then Read;
        until identype<>InProcess;
      end;

procedure Error(error:ErrorTypes);
```

```
 Print error message and terminate }
{G:9999,#bracket,ErrorTypes,line,MaxProc,#output,stack,StackDepth }

  begin
    write(output,'***** Error at',line:5,' ***** : ');
    case error of
NoProgram:
      writeln(output,'File does not begin with "program"');
Redefinition:
      writeln(output,'Procedure defined twice at same scope');
TooManyProcs:
      writeln(output,'Too many procedures, max',MaxProc:4);
Misplaced:
      writeln(output,'Misplaced reserved word');
TooDeep:
      writeln(output,'Too many nested blocks, max',StackDepth:3);
LostEnds:
      writeln(output,'End-of-file -- missing END''s?');
LostPeriod:
      writeln(output,'Unmatched END''s or missing EOF period');
      end;
    if bracket>1 then writeln(output,'Unterminated blocks:');
    while bracket>1 do begin {Print line #'s of unmatched BEGIN/CASE's}
      bracket := bracket-1;
      writeln(output,stack[bracket]:19);
      end;
    goto 9999;
    end;

unction FormatIdent(var ident:IdentString;
var identcases:IdentSet): IdentPtr;
 Restore upper-case for printing.  Pointers are used because the }
 result of a function must be either ordinal or pointer.         }
G:heapident,IdentLen,IdentPtr,IdentRange,IdentSet,IdentString,Offset }

 var j: IdentRange;
 begin
   FormatIdent := heapident;
   heapident^  := ident;
   for j := 1 to IdentLen do if j IN identcases then
    heapident^[j]:=chr(ord(heapident^[j])-Offset);
   end;

rocedure PrinTable(callstable:Boolean);
 Formatted output of collected reference data.  If several        }
 procedures have the same name, they appear in order of definition }
G:Boolean,!FormatIdent,IdentLen,LineWidth,list,#output, }
 procnum,ProcRange,ProcSet,prog,sortlist                 }

 const
   IdentWidth = 18 { IdentLen+2 ( two spaces before ident) };
   Indent     = 38 { continuation indentation: IdentLen+22 };

 var
   j          : 1..LineWidth;
   proc, ref  : ProcRange;
   refset     : ProcSet;

 begin
   writeln(output);
   writeln(output);
   write(output,'  def   body  level  Table of ');
   if callstable then write(output,'calls')
    else write(output,'callers');
   writeln(output,' for ',prog);
   writeln(output);
   for proc:=1 to procnum do with list[sortlist[proc]]^ do
    if callstable AND (calls<>[]) OR NOT callstable AND
     ((callers<>[]) OR (level>0)) then begin { Include each procedure }
    { if it called or was called, but include all user-defineds      }
    { in the table of callers in order to find never-useds           }
     j:=Indent;
     if defline=0 then write(output,' ':12) else begin (Non-intrinsics)
       write(output,defline:5);
       case bodyline of
:
         write(output,'  none?') (Body of forward procedure not found);
1:
         write(output,' extern');
axInt:
         write(output,' formal');
therwise
         write(output,bodyline:7);
         end;
       end;
     write(output,level:7,'   ',FormatIdent(name,namecases)^,':');
     if callstable then refset:=calls else refset:=callers;
     for ref:=1 to procnum do if sortlist[ref] IN refset then begin
       if IdentWidth>LineWidth-j then begin { No room left on line }
         writeln(output);
         write(output,':':Indent);
         j:=Indent;
         end;
       with list[sortlist[ref]]^ do write(output,'  ',
        FormatIdent(name,namecases)^);
       j:=j+IdentWidth;
       end;
     writeln(output);
     end;
   end;

nction FindProc(var proc: ProcRange): Boolean;
 Set arg to list[] element that points to the  }
 ProcDesc for ident. If none, result is false. }
:Boolean,ident,list,procnum,ProcRange         }

 begin if list[1]=NIL then FindProc:=false else begin
   proc:=procnum;
   while ((list[proc]^.name<>ident) OR (list[proc]^.scope=0)) AND
    (proc>1) do proc:=proc-1;
   FindProc:=true;
   with list[proc]^ do if(scope<>1)OR(name<>ident)then FindProc:=false;
   end end;

ocedure AddProc;
Add a ProcDesc for ident }
:block,!Error,!FindProc,ident,identcases, }
line,#list,MaxProc,#procnum,ProcRange     }
```



```
   var proc:ProcRange;
   begin
     if FindProc(proc) then if list[proc]^.level=block
      then Error(Redefinition);
     if procnum=MaxProc then Error(TooManyProcs);
     if list[1]<>NIL then procnum:=procnum+1;
     new(list[procnum]);
     with list[procnum]^ do begin { Initialize }
       name      := ident;
       namecases := identcases;
       level     := block;
       scope     := 1;
       defline   := line;
       bodyline  := 0;
       calls     := [];
       callers   := [];
       end;
     end;

 procedure AddIntrinsics;
 { Add the pre-defined procedures to the procedure list }
 {G:BlankIdent,#ident,#identcases,IdentRange,#line }

   const
     ConstLen = 53 { Length of the intrinsics-definition constants };
     Const1   = 'abs arctan chr cos dispose eof eoln exp get ln new o-';
     Const2   = 'dd ord pack page pred put read readln reset rewrite -';
     Const3   = 'round sin sqr sqrt succ trunc unpack write writeln - ';
 {}  Const4   = 'append close halt linepos mark maxpos open overprint-';
 {}  Const5   = ' position prompt readdir release seek writedir -     ';
   type
     ConstRange = 1..ConstLen;
     ConString  = packed array[ConstRange] of Char;
   var k:IdentRange;

  procedure AddIntrinsics(names:ConString);
  { Do the real work of the procedure.  Necessary since the intrinsics }
  { definition constant is sectioned - inner proc is called for each.  }
  {G:!AddProc,BlankIdent,ConString,ConstRange,#ident,#k }

    var j:ConstRange;
    begin
      j:=1;
      repeat
        if names[j]=' ' then begin { Add procedure }
          AddProc;
          j        := j+1;
          k        := 1;
          ident := BlankIdent;
          end
         else if names()'-' then begin { Read next char }
          ident[k] := names[j];
          k           := k+1;
          j           := j+1;
          end;
       until names[j]='-';
      end;

  begin { AddIntrinsics (outer) }
    ident        := BlankIdent;
    identcases := [];
    line         := 0;
    k            := 1;
    AddIntrinsics(Const1);
    AddIntrinsics(Const2);
    AddIntrinsics(Const3);
 {} AddIntrinsics(Const4);
 {} AddIntrinsics(Const5);
    line  := 1;
    end;

 procedure ProcessBlock;
 { Process a procedure, function, or program block }
 {G:!AddProc,#block,ExtIdent,!FindProc,!FormatIdent,ident,identcases, }
 { identype,line,list,LowProcRange,#output,#paren,procnum,ProcRange   }

   var
     proc                : ProcRange;
     current,localroot : LowProcRange;

   procedure ScanArguments;
   { Read arguments, checking for scope occlusions and formal procs }
   {G:!AddProc,block,!Error,!FindProc,!FormatIdent,ident,identcases, }
   { identype,line,list,#output,#paren,#proc,!ReadIdent              }

     begin { ScanArguments }
       paren:=0 { Should be anyway, but make sure };
       ReadIdent;
       while paren>0 do begin { Inside argument list }
         if identype=Other then if FindProc(proc)
          then list[proc]^.scope:=block
          else else if identype<>Def then Error(Misplaced)
          else begin { Formal procedure/function }
           ReadIdent;
           writeln(output,line:5,'  ':block*2,
            FormatIdent(ident,identcases)^);
           writeln(output,line:5,'  ':(block+1)*2,'formal');
           AddProc;
           list[procnum]^.bodyline:=MaxInt;
           end;
         ReadIdent;
         if paren>1 then repeat ReadIdent until paren<2;
         end;
       end;

   procedure ScanDefs;
   { Read definitions, checking for scope occlusions and local procs }
   {G:block,!FindProc,!FormatIdent,ident,identcases,identype,line, }
   { #list,#output,#proc,!ProcessBlock,!ReadIdent                  }

     begin while (identype<>Open) AND (identype<>Directive) do begin
       if identype=Other then if FindProc(proc)
        then list[proc]^.scope:=block
        else else if identype=Def then begin { Local procedure }
         ReadIdent;
         writeln(output,line:5,'  ':block*2,
          FormatIdent(ident,identcases)^);
```



```
         ProcessBlock;
         end;
       ReadIdent;
       end end;

   procedure ScanBody;
   { Check body for references to procedures }
   {G:bracket,current,!Error,!FindProc,#identype,input,line,list, }
   { #proc,!Read,!ReadIdent,                                       }

     procedure Push;
     { Stack a 'begin' or 'case' statement-bracket line-number }
     {G:#bracket,!Error,line,#stack,StackDepth }

       begin
         stack[bracket]:=line;
         if bracket=StackDepth then Error(TooDeep);
         bracket:=bracket+1;
         end;

     procedure Pop;
     {G:block,#bracket,!Error,input }

       begin if (input^='.') AND ((bracket>2) OR (block>1))
        then Error(LostEnds) else bracket:=bracket-1 end;

     begin { ScanBody }
       list[current]^.bodyline:=line;
       Push;
       repeat
         ReadIdent;
         case identype of
 Def,Directive:
            Error(Misplaced);
 Open,CaseOpen:
            Push;
 Close:
            Pop;
 Other:
            begin { Possible reference or assignment to a function }
              while input^=' ' do Read;
              if input^=':' then begin ( ':=' possible )
                Read;
                 if input^='=' then identype:=Def ( Assignment );
              end;
            if identype=Other then if FindProc(proc) then begin { Ref }
              with list[current]^ do calls:=calls+[proc];
              with list[proc]^ do callers:=callers+[current];
              end;
            end;
         end;
       until bracket=1;
     end;

 procedure DeleteDefs;
 { Set local procedures out-of-scope and re-instate occluded ones }
 {G:block,list,localroot,procnum,ProcRange }

   var proc:ProcRange;
   begin
     if localroot<procnum then for proc:=localroot+1 to procnum do
      list[proc]^.scope:=0;
     for proc:=localroot downto 1 do
      if list[proc]^.scope=block then list[proc]^.scope:=1;
     end;

 begin { ProcessBlock }
   current := 0;
   if FindProc(proc) then with list[proc]^ do
    if (level=block) AND (bodyline=0)
     then current:=proc { Body for a forward-declared procedure };
   if current=0 then begin { Add new procedure }
     AddProc;
     current:=procnum;
     end;
   localroot:=procnum;
   block:=block+1;
   ScanArguments;
   ScanDefs;
   writeln(output,line:5,'  ':block*2,FormatIdent(ident,identcases)^);
   if identype=Open then ScanBody
    else if ident=ExtIdent then list[current]^.bodyline:=-1;
   DeleteDefs;
   block:=block-1;
   end;

 procedure Sort;
 { Since there are so few procedures to sort, there }
 { is no need for a complicated algorithm           }
 {G:list,procnum,ProcRange,#sortlist }

   var
     proc,j,k : ProcRange;
     status   : (InProcess,Finished);

   begin
     sortlist[1]:=1;
     for proc:=2 to procnum do begin
       k:=proc DIV 2;
       status:=InProcess;
       with list[proc]^ do
        if list[sortlist[k]]^.name>name then repeat
         if k=1 then status:=Finished
          else if list[sortlist[k-1]]^.name>name then k:=k-1
          else status:=Finished
         until status=Finished
        else repeat
         if k=proc then status:=Finished
          else if list[sortlist[k]]^.name<=name then k:=k+1
          else status:=Finished
         until status=Finished;
       for j:=proc downto k+1 do sortlist[j]:=sortlist[j-1];
       sortlist[k]:=proc;
       end;
     end;

 procedure Read;
 { Check each char from input for end-of-line }
 {G:#input,#line }

   begin
     get(input);
     if eoln(input) then line:=line+1;
     end;

 begin { Pref }
 {}Exec(GoTime,time);
 {}with time do begin { Save start time }
 {}  sec := days*86400+hours*3600+minutes*60+secs;
 {}  mil := millisecs;
 {}  end;
   alpha      := ['A','B','C','D','E','F','G','H','I','J','K','L','M',
                  'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'];
   alphadigit := alpha+['a','b','c','d','e','f','g','h','i','j','k','l',
                        'm','n','o','p','q','r','s','t','u','v','w','x',
                        'y','z','0'..'9'];
   list[1]     := NIL;
   paren       := 0;
   block       := 0;
   bracket     := 1;
   line        := 1;
   procnum     := 1;
   new(heapident);
   if Intrinsics then AddIntrinsics;
   ReadIdent;
   if ident<>ProgIdent then Error(NoProgram);
   ReadIdent;
   prog:=FormatIdent(ident,identcases)^;
   writeln(output,' line  Table of definitions for ',prog);
   writeln(output);
   ProcessBlock { Phase 1 - Process input };
   if input^<>'.' then Error(LostPeriod);
   Sort { Phase 2 };
   If CallsTable   then PrinTable(true)  { Phase 3A };
   If CallersTable then PrinTable(false) { Phase 3B };
 {}writeln(output);
 {}GetInfo(info);
 {}Exec(GoTime,time);
 {}with info,time do begin { Print statistics }
 {}  sec := days*86400+hours*3600+minutes*60+secs-sec;
 {}  mil := millisecs-mil;
 {}  if mil<0 then begin { Correct for borrow from milliseconds }
 {}    mil := mil+100;
 {}    sec := sec-1;
 {}    end;
 {}  write(output,'Heap = ',initoh-toh:1,' words.  Time = ',sec:1,'.');
 {}  if mil<10 then write(output,'0');
 {}  writeln(output,mil:1,' seconds.');
 {}  end;
   page(output);
   9999:
     end.
```

PUG

Dear Rich:

The software tools section of Pascal News is extremely useful. We have implemented Prose on the HP 3000 and we enjoy using Prose to do our text formatting.

This letter includes one enhancement to Prose and one bug-fix. The enhancement provides a new terminal-type: DIABLO. This terminal-type provides for proportional spacing on DIABLO terminals. The changes are as follows:

Lines 167 to 173 become:

```
    { THE FOLLOWING ARE NOT DIRECTIVES, BUT IT IS CONVENIENT
    { TO INCLUDE THEM IN THIS TABLE.

            AST,                { ASCII TERMINAL                }
            LPT,                { LINE PRINTER                  }
            AJT,                { ANDERSON/JACOBSON TERMINAL    }
            DIA,                { DIABLO TERMINAL               }
            ILT);               { ILLEGAL                       }
```

Lines 789 to 793 become:

```
            CASE TERMINALTYPE OF
             AJT,
             DIA,
             AST: WRITE1(CR);
             LPT: BEGIN
                      WRITELN(OUTPUT);
                      CARRIAGECONTROL:=PLUS
                  END
            END
```

Lines 824 to 825 become:

```
              END { IF TERMINALTYPE = AJT }
            ELSE
            IF TERMINALTYPE = DIA THEN
              BEGIN
                X2 := 0;
                FOR X1 := 1 TO LEN DO
```

```
            WITH STR[X] DO
              IF C <> BLANK THEN
                BEGIN
                  IF X2 <> 0 THEN
                    BEGIN
                      IF (X2 MOD CHARWIDTH = 0) THEN
                        FOR X3 := 1 TO (X2 DIV CHARWIDTH) DO
                          WRITE(BLANK)
                      ELSE
                        BEGIN
                          FOR X3 := 1 TO (X2 DIV CHARWIDTH) DO
                            WRITE(BLANK);
                          X2 := X2 MOD CHARWIDTH;
                          WRITE(ESC);
                          WRITE(THREE);
                          FOR X3 := 1 TO X2 DO
                            WRITE(BLANK);
                          WRITE(ESC);
                          WRITE(FOUR);
                        END
                    END;
                  X2 := 0;
                  WRITE(C)
                END
              ELSE  X2 := X2 + NB1
          END
        ELSE
        FOR X1 := 1 TO LEN DO

Lines 1852 to 1860 become:

        AJT,
        DIA: BEGIN
               WHILE INCHAR = BLANK DO
                 NEXTCH;
               CHARWIDTH := NUMBER(10, -1, 0, INFINITY, 1013);
               IF NOT (CHARWIDTH IN [10, 12]) THEN
                 BEGIN
                   ERROR(1013);
                   CHARWIDTH := 10
                 END;
               IF (TERMINALTYPE = DIA) AND (CHARWIDTH = 12) THEN
                 BEGIN
                   WRITE(ESC);     {Write out the HMI}
                   WRITE(US);
                   WRITE(FF);
                 END;
               CHARWIDTH := 60 DIV CHARWIDTH;
               OUTLINE[1].NB1 := LEFTMARGIN * CHARWIDTH
             END

Lines 3439 to 3440 become:

  IF ERRORS THEN WRITELN(' PROSE ERRORS DETECTED.');
  IF (TERMINALTYPE = DIA) AND (CHARWIDTH = 5) THEN
    BEGIN    {RESET PITCH}
      WRITE(ESC);
      WRITE(S);
    END
  END.  { PROSE }
```

The version of Prose published in PN # 15 contains a bug concerning index entries. If an index entry is underlined, Prose starts referencing the NIL pointer. The problem is that the function UPPER returns an incorrect value for underlined characters. A new UPPER function is introduced in the SORT procedure.

```
Lines 2169 to 2170 become:

        X1          : INTEGER;  { GENERAL INDEX VARIABLE       }

{           UPPER - SPECIAL VERSION OF UPPER.  DOES NOT RETURN
 *                  UNDERLINED CHARACTERS.
 *
 *           PARM CH = CHARACTER TO CONVERT TO UPPER CASE.
 }

      FUNCTION UPPER( CH : ASCIIX) : ASCIIX;
      BEGIN { UPPER }
        IF ODD(CH DIV 128) THEN
          CH := CH - 128;
        IF CLASS(CH).LETTER THEN
          IF CH >= SMALLA THEN
            UPPER := CH - 32
          ELSE
            UPPER := CH
        ELSE
          UPPER := CH;
      END {UPPER};


      BEGIN  { SORT }
```

I encourage all Prose users to send their changes to Pascal News. With such an excellent tool it would be unfortunate if widely varying versions were to start appearing.

Yours truly,
David J. Greer

The Use of Generic Capsules with the University of Minnesota Pascal 6000 Compiler

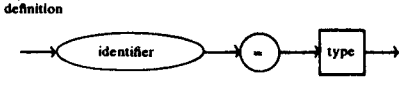
by Frank L. Friedman
Alessio Giacomucci
Carol A. Ginsberg
Anita Girton
Temple University

I. INTRODUCTION

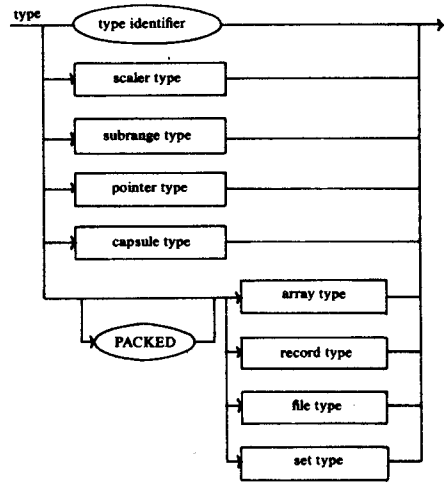
This document contains a description of a data type abstraction facility, a *capsule*, that has been implemented as an extension to the University of Minnesota Pascal 6000 Series compiler. The facility provides an encapsulation that establishes a static scope of identifiers with controlled visability. Data objects and a set of operations on these objects may be enclosed. The document is intended to provide sufficient information for those who wish to use the general capsule facility and library. A more complete description of capsules may be found in the paper "Capsules: A Data Abstraction Facility for Pascal," CIS-TR 81-01, Temple University C & IN SC Department Technical Report.

II. WHAT IS A CAPSULE?

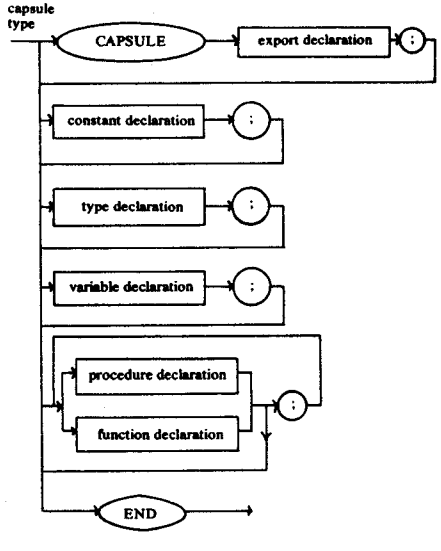
A capsule is an additional Pascal type which is syntactically similar in structure to the Pascal *record*. The syntax diagrams for the Pascal type definition (with the capsule added) may be specified as



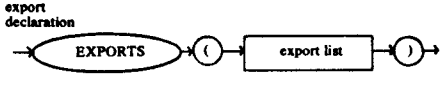
Department of Computer and Information Sciences, Computer Users Document 81-01, February, 1981, Rev. 1, September, 1981, Rev. 2, December, 1981



The capsule type is defined by the diagram



with the export declaration defined as



The export list is a list of variable, procedure and function identifiers which may be referenced outside the scope of the capsule. All protection of the data objects encapsulated in the capsule is provided at compile time. Thus, if capstype is a capsule, and the variable X is declared to be of type capstype, then all external references to identifiers, id, appearing in the export list for capstype must be of the form

X$id

Exported variables are read only, and identifiers not appearing in the export list may not be referenced outside the scope of the capsule. There is no explicit import facility, such as provided in Modula and Euclid.

The Pascal scope rules for capsules are the same as the rules for all other Pascal objects. Only a single copy of the operations (procedures and functions) defined within a capsule is created, regardless of the number of variables declared to be of the capsule type. When a procedure (or function) containing the declaration of a capsule-type variable is called and the variable declaration is elaborated, the capsule's global variables are placed on the runtime stack as a record. This record remains on the stack as long as the called procedure (function) remains active. Operations on the abstract objects are thus performed via calls of the appropriate capsule procedures or functions.

An example of a capsule in parameterized (generic) form is shown in Figure 1. An illustration of the use of this capsule is shown in Figure 1.

```
                {A non-recursive expression parser}

                type

A.                  $G ('capstk'/'capsall', charstack20,20,char);
                    .
                    .
                var
                    stack:  charstack20;
B.                  .
                    .
C.              begin     {initialize} stack$init;
                    .
                    .
                {   stack$push (cursym);
D.                  .
                    .
                    stack$pop (rightoperand);
                    stack$pop (operator);
                    stack$pop (leftoperand)
                    .
                    .
                end {parser};
```

Figure 1:

Use of a simple stack capsule

```
E.  capstk

F.  (pname, psize, ptype)        {list of capsule parameters}

    pname = capsule

    {stack capsule definition (in generic form)
     *
     *
     *  parameters:
     *  pname  - name of capsule
     *  psize  - number of elements in the stack
     *  ptype  - base type of stack array
     }
```

```
G.  exports (pop, push, init);  {exported identifiers}

    type    stackpointer = 0..psize;

H.  var     {global capsule variables}
            a:  array  [1..psize] of ptype;  {stack}
            top:  stackpointer;  {pointer to top of stack}

    procedure pop (var item:  ptype);
        {pop an item of the stack and save in item}
        ...
        end (pop)

    procedure push (item:  ptype);
        {push item onto stack}
        ...
        end (push);

I.  procedure init;
        {performs required initialization of global objects}
        begin
            top = 0
        end  {init};

J.  procedure print;
        {print out the data}
        begin
            (*$Y IF('PTYPE'='REAL')THEN*)
            writeln  (  datavalue:5:2)

            ( *$Y ELSEIF('PTYPE'='INTEGER')*)
            writeln  (  datavalue:5);

            {*$Y ELSE insert the next line to inform user of errors}
            to the user:  ptype must be type integer or real, only.

            ( *$Y ENDIF*)

        end (print) ;

    end  (generic form of stack) ;
```

Figure 2:

Stack capsule: generic form

The major features of the capsule facility are indicated by the letters A-H in the left hand margins of these figures. These features are discussed next.

A. **Generic (Parameterized) reference:** Generic references in a Pascal program are processed by the Generics Preprocessor (see Section III). This program searches a library of generic capsules (capsall in this case) for the named capsule record (capstk), and copies the capsule text into the program, substituting the designated arguments (charstack20, 20 and char) for the generic parameters (pname, psize and ptype) listed in the capsule header (see line F.). The syntax for specifying a reference to a generic capsule is patterned after the syntax for the INCLUDE facility provided by the Minnesota Pascal Compiler.+

B. **Instantiation of a data element of type charstack20 all about one stack:** This creates an instance of the capsule: a copy of the global variables of the capsule will be placed on the run-time stack when this declaration is elaborated during execution.

C. **Call to initialization:** The global capsule variable, top, will be initialized to zero when this call is executed.

+See the University of Minnesota Pascal 6000 Release 3 document.

D. **References to exported identifiers:** An exported identifier is referenced by prefixing it with the capsule name followed by a dollar sign.

E. Capstk is the name of the capsule record as referenced in the generic statement (see A.).

F. **Capsule Parameter List:** Generic arguments (charstack20, 20 and char in this case) are substituted for the parameters (pname, psize and ptype) each time the capsule is referenced in a generic statement. As illustrated in the capsule header statement in the line following the parameter list, the use of the parameter pname permits the user to assign different names to each different stack capsule that is needed.

G. **Export list:** The export list is a list of all capsule identifiers (variables, procedures, functions) that may be referenced from outside the capsule.

H. **Declaration of global (permanent) objects:** For each variable declared to be of the capsule type, a copy of these objects is placed on the run-time stack.

I. **The initialization procedure:** If the initialization of global capsule data is required, such a procedure must be called explicitly by the user for each declared instance of the capsule.

The examples in Figures 1 and 2 also illustrate some of the shortcomings of the current capsule implementation. For example, there is no provision for the automatic execution of initialization statements, such as provided in Concurrent Pascal. There is also no provision for the direct specification of variable initialization in a declaration, a feature that is provided by Ada, Euclid, and CLU. Rather, any initialization required for the encapsulated data object must be done via an explicit reference to an initialization procedure (such as init) defined within the capsule.

III. GENERIC CAPSULE PREPROCESSOR

A. Introduction

The Generic Capsule Preprocessor (GCP) is a program that may be used to allow a programmer to insert Pascal source text anywhere in a Pascal source program. The GCP is patterned after the Pascal INCLUDE facility (see the document Pascal 6000 Release 3) and is used primarily for the insertion of Generic Capsules into the type declaration section of a user program, procedure, or function.

B. Use of the GCP

1. To use the GCP, the programmer must first create a capsule library either in the form of a sequential file of capsules [with each capsule separated by an end-of-record (7/8/9 or *EOR], or a user library file of capsules (using the CDC Modify source library maintenance system).

   If the sequential file approach is taken, the file must appear as shown in Figure 3. Such a file may easily be created and maintained using SENATOR (see TUCA documents E601 or E602).† For large collections of capsules, the CDC Modify system is recommended for creation and maintenance of the capsule library (see the CDC Manual on Modify for additional details).

   In Figure 3, the first line of each record indicates the record name. The second line contains the list of parameters ($n_1 \leq 9$) to be replaced when the capsule is copied from the library. If there are no parameters, this line may be omitted.

†Temple University Computer Activity, introductory and advanced level documents on interactive computing.


```
recnam1
(par1, par2, ... parn1)
{
 capsule body
*EOR
 recnam2
 (par1, par2, .... parn2)
{
 capsule body
*EOR
        .
        .
        .
```

Figure 3:

Structure of a Sequential File of Capsules

2. Capsules may be retrieved from a capsule library (and copied into a Pascal module) through the use of the Pascal G compiler option:

   $G('recnam'/'libfilnam')

   or

   $G('recnam'/'libfilnam', arg1, arg'i2, ..., arg_n)

   where
   - recnam — the name of the capsule record to be inserted
   - libfilnam — the name of the capsule library file containing the record
   - arg_1, ..., arg_n — the actual parameters to be substituted (via text string substitution) for the dummy parameters in the definition of the capsule record.

   Remember that Pascal compiler options must be inserted inside a comment, and may contain no blanks.

3. Example

The generic stack capsule shown in Figure 2 contains three parameters, pname, psize, ptype which can be used to specify the capsule name, the size of the array to represent the stack, and the type of the information to be stored in each element of the stack.

When encountered by the GCP, the statement

(*$G('capstk'/'capsall', charstack20, 20, char)*)

causes an instance of the stack capsule to be copied into the user's text at the point of reference. During the copy, each occurrence of the parameters pname, psize and ptype would be replaced by the corresponding arguments, charstack20, 20, and char. The result, in this case, would be a capsule named charstack20 which uses a 20-element array of elements of type char. Given this capsule definition, variables such as x,y,z declared as

var x,y,z; charstack20;

would represent character stacks of size 20 which could be manipulated using the pop, push, and init functions specified in the capsule.

The reference

(*$G('capstk'/'capsall',instack1000,1000,integer)*)

could be used to establish a capsule definition for a stack consisting of an array of 100 integers. The delcaration

var w,z: intstack1000;

would establish variables w and z each representing integer stacks of size 1000.

C. Restrictions and Other Comments

1. A generic reference $G... may not be the first statement of an input program, since a program statement is expected here.
2. Only one capsule library file may be accessed at a time.
3. If no substitution is desired for a particular parameter, par_i, in a capsule record, use a null argument (indicated by consecutive commas) in the position corresponding to par_i. Thus

   $G('capstk'/'capsall',charstack20,,char)

   would have the effect of leaving psize untouched when the stack capsule is copied into the user program.
4. No capsule parameter (appearing in a generic capsule record) may exceed 10 characters in length.
5. A maximum of 9 parameters is allowed for a given generic capsule.

D. Use of conditional inclusion within a capsule

1. a. Any conditional statement may be included within a generic capsule which is part of a capsule library. There must be at least one *capsule* parameter which will be the basis for testing the condition. A conditional statement must never precede the capsule parameter statement, but it must procede the EOF marker of the capsule within the library. (Refer to Fig. 2, the stack capsule).

   b. The permissible conditional statements may begin with only one of the following: 'IF', 'ELSE', 'ELSEIF', 'ENDIF'. One 'endif' statement is required for each 'if', statement. No 'elseif' statement may logically follow an 'else' statement.

   c. The only relational operators permitted are as follows:

      = < > <= >= < >

   d. No blanks are permitted in the formal part of the statement, except the one which follows the 'Y'

as noted in part 2 below. Alternatively, the space may be used to note the level number of nested statements, for readability.

2. Conditional inclusion of text within a generic capsule may be instituted through use of the PASCAL Y compiler option:

   $Y keyword

   or

   $Y keyword ('paramname'op'paramvalue')

   or

   $Y3keyword

   or

   $Y3keyword ('paramname'op'paramvalue')

   where
   - keyword — the word IF or ELSE or ELSEIF or ENDIF.
   - paramname — a parameter name, exactly as it appears in the capsule parameter list.
   - op — one relational operator chosen from the set:

     =, < >, <, >, < =, > =

     (Note: ONLY = and < > may be used in comparing alphabetic operands).
   - paramvalue — a parameter value to be compared against the corresponding argument in the generic capsule call statement ($G statement).

Remember that the Pascal compiler option must be inserted within comment markers, and may contain no blanks except as specifically stated. User comments may immediately precede the closing comment marker.

3. Remarks

The generic stack processing capsule in Figure 2 provides an example of the use of conditional inclusion. As shown, the condition inclusion feature was used to determine the type of data to be printed by procedure "print." The feature may also be used to restrict the use of a capsule based on a capsule user's knowledge of parameter values. At times, the feature may be utilized to insert a variety of comments in the output program, depending on substituted parameter values.

The form of the conditional statement which includes a numerical digit following the 'Y' may be used to help distinguish among IF-THEN-ELSE statements which are nested. For example:

```
(*$Y1IF('PTYPE'='REAL')THEN*)
     .
     .
     .
   (*$Y2IF('PCOLORS' '9')*)
          .
          .
          .
   (*$Y2ELSE*)
          .
          .
          .
   (*$Y2ENDIF of color checking*)

(*$Y1ELSE*)
     .
     .
     .
(*$Y1ENDIF*)
```

ACKNOWLEDGEMENT

The authors would like to thank Professor Giorgio P. Ingargiola of the Temple Computer Science Department for many helpful comments and suggestions concerning the design of the capsule facility.

BIBLIOGRAPHY

[Barnard 78] Barnard, David T., W.D. Elliott and David H. Thompson, *"Euclid and Modula," SIGPLAN Notices* (13,3), pp. 70-84, March, 1978.

[Brand 78] Brand, D. "A Note on Data Abstractions," *SIGPLAN Notices* (13,1), pp. 21-24, January 1978.

[Brinch-Hansen 75] Brinch-Hansen, P., "The Programming Language Concurrent Pascal," *IEE Transactions of Software Engineering* (1,2), June 1975.

[Chand 78] Chand, D.R. and S.B. Yadav, "On the Applications of Data Abstraction Facilities," *Proceedings of the 1978 ACM Annual Conference*, December, 1978, pp. 639-645.

[Chang 78] Chang, E., N. Kaden and W. Elliot, "Abstract Data Types in EUCLID," *SIGPLAN Notices* (13,3) pp. 34-40 (March, 1978).

[Coleman 78] Coleman, Derek, *A Structured Approach to Data*, The MacMillan Press Limited, London, England, 1978.

[Dahl 72] Dahl, O.J. and C.A.R. Hoare, "Hierarchical Program Structures," in *Structured Programming* by O.J. Dahl, E.W. Dijkstra, and C.A.R. Hoare, Academic Press, 1972.

[Dijkstra 72] Dijkstra, E.W., "Notes on Structured Programming," in *Structured Programming*, by O.J. Dahl, E.W. Dijkstra, and C.A.R. Hoare, Academic Press, 1972.

[DoD 79] DoD, "Preliminary Ada Reference Manual," *SIGPLAN Notices* (14, 6A), June, 1979.

[Friedman 79] Friedman, Frank L. and Judith A. Stebulis, "An Undergraduate Compiler Laboratory," *SIGCSE Bulletin* (11, 1), February, 1979, pp. 28-36.

[Girton 81] Girton, Anita, "A Generic Capsule Preprocessor" (in progress).

[Halstead 77] Halstead, M.H., "Elements of Software Science," *Elsevier Computer Science Library*, 1977.

[Horning 75] Horning, J., "Some Desirable Properties of Data Abstraction Facilities," *Proceedings of the Conference on Data: Abstraction, Definition and Structure, SIGPLAN Notices* (8, 2), March, 1976, pp. 60-62.

[Ichbiah 79] Ichbiah, J.D., et. al., "Rationale for the Design of the Ada Programming Language," *SIGPLAN Notices* (14, 6B), June, 1979.

[Lampson 77] Lampson, B., et. al., "Report on the

Programming Language Euclid," *SIGPLAN Notices* (12, 2), February, 1977.

[Linden 76] Linden, Theodore, "The Use of Abstract Data Types to Simplify Program Modification," *SIGPLAN Notices (8, 2), pp. 12-23, March, 1976.*

[Liskov 74] Liskov, B., "A Note on CLU," in CLU Design Notes, Project MAC, MIT, Cambridge, MA., 1974.

[Liskov 77] Liskov, B., Snyder, A., Atkinson, R. and Schaffert, C. "Abstraction Mechanisms in CLU," *Comm. ACM*, Vol. 20, No. 8, pp. 564-576, 1977.

[Liskov 77] Liskov, B., et. al. "CLU Reference Manual," *Computation Structures Group Memo No. 161*, Laboratory for Computer Science, MIT, Cambridge, MA., 1978.

[McCabe 76] McCabe, T.J., "A Complexity Measure," *IEEE Transitions on Software Engineering*, SE-2 (4), pp. 308-320, 1976.

[McCall 80] McCall, J.L., and M.T. Matsumoto, "Software Quality Metrics Enhancements," *General Electric Company/Rome Air Development Center Final Technical Report* RADC-TR-80-109, Volume I, April, 1980.

[Mickel 79] Mickel, Andrew B., et. al., *Pascal 6000 Release 3 Document*, University of Minnesota, Minneapolis, MN, January, 1979.

[Palme 76] Palme, J., "New Feature for Module Protection in SIMULA," *SIGPLAN Notices*, Vol. 11, No. 5, pp. 59-62, 1976.

[Shaw 78] Shaw, Mary, et. al., "Validating the Utility of Abstraction Techniques," *Proceedings of the ACM Conference*, Washington, D.C., pp. 106-110, December, 1978.

[Sheil 81] Sheil, B.A., "The Psychological Study of Programming," *ACM Computing Surveys* (13, 1), March, 1981, pp. 101-120.

[Venema 78] Venema, Ted and Jim des Rivieres, "Euclid and Pacal," *SIGPLAN Notices*, (13, 3), pp. 57-69, March, 1978.

[Yin 78] Yin, B.H., and I.W. Winchester, "The Establishment and Use of Measures to Evaluate the Quality of Software Designs," *Proceedings Software Quality Assurance Workshop: Functional and Performance Issues*, San Diego, November, 1978.

[Wegner 79] Wegner, Peter, "Programming with Ada: An Introduction by Means of Graduated Examples," *SIGPLAN Notices* (14, 12), pp. 1-46, December, 1979.

[Welsh 80] Welsh, Jim, and Michael McKeag, *Structured System Programming*, Prentice-Hall, 1980.

[Wirth 71] Wirth, N., "The Programming Language Pascal," *Acta Informatica* 1, pp. 35-63, 1971.

[Wirth 76] Wirth, N., *Algorithms + Data Structures = Programs*, Prentice-Hall, 1976.

[Wirth 77] Wirth, N., "Design and Implementation of Modula," *Software — Practice and Experience* (7), pp. 67-84, 1977.

Histogram Capsule
(from Dahl and Hoare — [Dahl 72])

```
HISTOGR
(PNAME,PNUMCATS,ITEMTYPE,PBNDTYPE)
PNAME = CAPSULE
(****************************************************************)

(********** HISTOGRAM CAPSULE **********)
```



```
(*
* TABULATES FREQUENCY DISTRIBUTION FOR A
* TIENTYPE RANDOM VARIABLE ON THE
* OPEN INTERVAL (-INF,+INF) WITH BOUNDARIES GIVEN BY THE ARRAY B
*    (-INF,B1),(B1,B2),(B2,B3)..(BN-1,BN),(BN,+INF)
*
* HISTOGRAM OPERATORS --
*       INIT - INITIALIZE BOUNDS ARRAY GIVEN THE BOUNDARIES
*              OF CATEGORIES (OF TYPE PBNDTYPE), AND CLEAR
*              FREQUENCY ACCUMULATOR ARRAY.
*       TABULATE - DETERMINE CATEGORY OF GIVEN ITENTYPE ITEM AND
*              INCREMENT FREQUENCY COUNT OF THAT CATEGORY
*       FREQUENCY - RETURN INTEGER FREQUENCY COUNT FOR GIVEN CATEGORY
*       PRINT - PRINT TABLE OF CATEGORIES,FREQUENCIES AND RELATIVE
*              FREQUENCIES.
*
* GENERIC PARAMETERS --
*      PNAME - NAME OF CAPSULE
*      PNUMCATS - NUMBER OF CATEGORIES (PARTITIONS) MINUS 1
*         THIS IS THE SAME AS THE NUMBER OF LOWER BOUNDS
*      ITENTYPE - TYPE OF VALUE BEING CLASSIFIED (A SCALAR).
*      PBNDTYPE - TYPE OF THE STRUCTURE HOLDING THE BOUNDARY VALUES
*
* ADAPTED FROM THE DAHL/HOARE PAPER 'HIERARCHICAL PROGRAM STRUCTURES
* IN THE DAHL/DIJKSTRA/HOARE TEXT ON STRUCTURED PROGRAMMING
*
*)

EXPORTS (TABULATE,FREQUENCY,PRINT,INIT);

CONST N = PNUMCATS;

TYPE RANGEZEROTON = 0..N;
     FREQARRAY = ARRAY[RANGEZEROTON] OF INTEGER;

(* GLOBAL CAPSULE VARIABLES *)

VAR B: PBNDTYPE; (*ARRAY OF LOWER BOUNDS FOR EACH CATEGORY 0..N -
                   B[0] = -INF = LOWER BOUND OF CATEGORY 0,
                   B[1] = LOWER BOUND OF CATEGORY 1,
                   B[N] = LOWER BOUND OF CATEGORY N*)
    FREQ: FREQARRAY; (*ARRAY OF FREQUENCIES, ONE FOR EACH OF
                   CATEGORIES 0..N*)
    TOTALCOUNT: INTEGER; (*TOTAL COUNT OF ITEMS PROCESSED*)


(***** PROCEDURE TABULATE *****)

PROCEDURE TABULATE(ITEM:ITENTYPE);
(*
 * DETERMINE CATEGORY [I] FOR ITEM AND INCREMENT FREQ[I] BY 1
 *
 * ARGUMENT DEFINITIONS ---
 *   INPUT ARGUMENTS
 *     ITEM - ITEM TO BE CATEGORIZED
 *)

LABEL 50;

VAR
    I,J: INTEGER;
BEGIN (*TABULATE*)
   FOR I := 1 TO N DO
      IF ITEM < B[I]
         THEN
            BEGIN
               J := I-1;
               FREQ[J] := FREQ[J]+1;
               GOTO 50
            END;
   (*ELSE INCREMENT LAST FREQUENCY CATEGORY*)
   FREQ[N] := FREQ[N]+1;
   50: TOTALCOUNT := TOTALCOUNT+1;
END(*TABULATE*);

(***** PROCEDURE PRINT *****)

PROCEDURE PRINT;
(*
 * PRINT A TABLE WITH CATEGORIES ON LEFT AND FREQUENCIES IN
 *   CENTER AND RELATIVE FREQUENCIES ON RIGHT.
 *
 * ARGUMENT DEFINITIONS -- (NONE)
 *
 * PROCEDURE EXPECTS TABLE HEADERS TO HAVE BEEN PRINTED ALREADY.
 * IT PRINTS A THREE-COLUMN TABLE WITH COLUMN HEADERS.
 *
 * EACH LINE OF THE TABLE APPEARS AS FOLLOWS -
 *)
(*  [LOWBOUND, HIBOUND)    FREQUENCY     RELATIVE FREQUENCY *)

CONST LPAR = '(';
      RPAR = ')';
      RBRACK = ']';
      COMMA = ',';
      LBRACK = '[';

VAR I: INTEGER;
BEGIN
   WRITE('0      CATEGORY RANGE                 ');
   WRITELN('FREQUENCY                RELATIVE FREQUENCY');
   WRITE('  ----  -- ------ -------');
   WRITELN('--  ----      ------------ -  ----------------------');
   WRITE(' ',LPAR,'        INF',COMMA,B[1]:10:2,RPAR);
   WRITE('          ',FREQ[0]:10 );
   WRITELN('             ',(FREQ[0]/TOTALCOUNT): 10: 4);
   FOR I:= 1 TO N-1 DO
      BEGIN
         WRITE(' ',LBRACK,B[I]:10:2,COMMA,B[I+1]:10:2,RPAR,'        ');
         WRITELN(FREQ[I]:10,
           '             ',(FREQ[I]/TOTALCOUNT):10:4)
      END; (*FOR I*)
   WRITE(' ',LBRACK,B[N]:10:2,COMMA,'       +INF',RPAR,'       ');
   WRITE(' ',LPAR,'       -INF',COMMA,B[1]:10,RPAR);
   WRITE('          ',FREQ[0]:10 );
```

```
    WRITELN('                    ',(FREQEC3/TOTALCOUNT):10:4 )#
    FOR I:= 1 TO N-1 DO
        BEGIN
            WRITE(' '+LBRACK+BE[3]:10+COMMA+BL[3]:10+RPAR+'          ')#
            WRITELN(FREQE[3]:10+
                    '          ',(FREQE[3]/TOTALCOUNT):10:4)
        END; (*FOR I*)
    WRITE(' '+LBRACK+BE[3]:10+COMMA+'      '+INF+RPAR+'          ')#
    WRITELN(FREQE[3]:10 +'          ',(FREQE[3]/TOTALCOUNT):10:4)#
    WRITELN('OTOTAL ITEMS PROCESSED ',TOTALCOUNT:3)#
END (*PRINT*)#

(*$$$$$ FUNCTION FREQUENCY *****)

FUNCTION FREQUENCY(I:RANGEZERUTON): INTEGER#
(*
 * RETURNS A FREQUENCY COUNT FOR CATEGORY I: 0<=I<=N
 *
 * ARGUMENT DEFINITIONS --
 *     INPUT ARGUMENTS
 *       I - INDEX OF FREQUENCY CATEGORY TO BE RETURNED
 *
 *)
BEGIN
    FREQUENCY := FREQE[I]
END (*FREQUENCY*)#


(*$$$$$ PROCEDURE INIT *****)

PROCEDURE INIT(EXTBOUNDARRAY:PBNTYPE)#
(*
 * INITIALIZE BOUNDARY ARRAY BE[0..N]# SET FREQ ARRAY AND
 *     TOTAL COUNT TO ZERO.
 *
 * ARGUMENT DEFINITIONS --
 *     INPUT ARGUMENTS
 *       EXTBOUNDARRAY - ARRAY OF BOUNDS THAT DEFINE THE CATEGORIES
 *)
VAR I: INTEGER#
BEGIN
    TOTALCOUNT := 0#
    FOR I := 1 TO N DO
        BE[I] := EXTBOUNDARRAY[I]#
    FOR I := 0 TO N DO
        FREQE[I] := 0#
END (*INIT*)#
END (*NAME (NESTOOKAH) CAPSULE*)#
```

Sort Capsule
(with conditional insertion directives)

```
(*SORT
(*NAME,*PTYPE,*PSUBRANGE,*PSTRUCTURE,*PDIRECTION,*PKEY)
*NAME = CAPSULE
(* SORT CAPSULE DEFINITION (IN GENERIC FORM)
 *
 * AUTHOR: CAROL A. GINSBERG
 * DATE COMPLETED: MARCH 1, 1982
 * LAST DATE MODIFIED: MARCH 5, 1982
 *
 * SORTS DATA IN ASCENDING ORDER (IF PDIRECTION = 'UP') OR
 * DESCENDING ORDER (IF PDIRECTION = 'DOWN') USING A SIMPLE
 * SELECTION SORT
 * PARAMETER DEFINITIONS --
 *     PNAME - NAME OF CAPSULE
 *     PTYPE - BASE TYPE OF ARRAY TO BE SORTED
 *     PSUBRANGE - TYPE (RANGE) OF INDEX TO ARRAY BEING SORTED
 *     PSTRUCTURE - INDICATES IF BASE TYPE IS 'SIMPLE' OR 'RECORD'
 *                  TYPE. IF 'SIMPLE' IS NOT DESIGNATED, THEN 'RECORD'
 *                  IS ASSUMED.
 *     PDIRECTION - INDICATION IF SORT IS TO BE ASCENDING OR DESCENDING
 *                  ORDER.
 *     PKEY - NAME OF RECORD KEYFIELD (REQUIRED IF ARRAY ELEMENTS
 *            ARE RECORDS)
 *
 *)
    EXPORTS (SORT)#

    TYPE DATTYPE = ARRAY[PSUBRANGE] OF PTYPE#

    (* THERE ARE NO GLOBAL VARIABLES REQUIRED FOR THIS
     * CAPSULE, BUT ONE DUMMY VARIABLE MUST BE DECLARED.
     *)
    VAR DUMMY: INTEGER#

PROCEDURE SORT (VAR X: DYNAMIC DATTYPE) FIRST,LAST:PSUBRANGE)#
(*
 * SORT DATA USING A SIMPLE SELECTION SORT
 *
 * DATA IS PLACED IN PDIRECTION ORDER
 *     (UP = ASCENDING) (DOWN = DESCENDING)
 *
 * ARGUMENT DEFINITIONS: --
 *     INPUT ARGUMENTS
 *       FIRST,LAST - LOWER AND UPPER LIMITS OF INDEX TO X ARRAY
 *       X - ARRAY TO BE SORTED
 *
 *)

LABEL 99#

TYPE
    INDEXTYPE = PSUBRANGE#

VAR I,J: INDEXTYPE# (*LOOP CONTROL VARIABLES*)
    TEMP: PTYPE# (*TEMPORARY VARIABLE FOR EXCHANGE*)
    IX: INDEXTYPE# (*INDEX OF LARGEST OR SMALLEST ITEM IN EACH SCAN*)

BEGIN (*SORT*)
(*VALIDATE NB*)
    IF FIRST >= LAST
        THEN
            BEGIN
                WRITELN ('OSORT CALLED WITH FIRST >= LAST')#
                WRITELN (' NO SORT PERFORMED.')#
                GOTO 99
            END# (* I) TO VALIDATE NB*)
(*SORT DATA*)
    FOR I := FIRST TO PRED(LAST) DO
        BEGIN (*FIND LARGEST (OR SMALLEST) ITEM BETWEEN X[I] AND X[N]*)
            IX := I#
            FOR J:=SUCC(I) TO LAST DO
(*$Y1IF('PDIRECTION'='UP')THEN *)
(*$Y2IF('PSTRUCTURE'='SIMPLE')THEN *)
                IF X[J] < X[IX]
                    THEN
                        IX := J#
(*$Y2ELSE*)
                IF X[J].PKEY < X[IX].PKEY
                    THEN
                        IX := J#
(*$Y2ENDIF*)
(*$Y1ELSEIF('PDIRECTION'='DOWN')THEN*)
(*$Y2IF('PSTRUCTURE'='SIMPLE')THEN*)
                IF X[J] > X[IX]
                    THEN
                        IX := J#
(*$Y2ELSE*)
                IF X[J].PKEY>X[IX].PKEY
                    THEN
                        IX := J#
(*$Y2ENDIF*)
(*$Y1ELSE*)
                BEGIN
                    WRITELN('INCORRECT DIRECTION INDICATOR FOR SORT')#
                    WRITELN(' RECOMPILE CODE WITH CORRECTED SORT')#
                    WRITELN(' EXECUTION TERMINATED.')#
                    GOTO 99  (*EQUIVALENT TO 'HALT'*)
                END; (*INNER LOOP*)
(*$Y1ENDIF*)

(*EXCHANGE ELEMENTS I AND IX*)
            TEMP := X[IX]#
            X[IX] := X[I]#
            X[I] := TEMP
        END; (*OUTER LOOP*)
    99:
END (*SORT*)#
END (*SORT CAPSULE*)#
```

Stack Capsule

```
CAPSTA
(PNAME,*PSIZE,PTYPE)
PNAME = CAPSULE
(**************************************************************************)

(********** STACK CAPSULE **********

* PROVIDES COMPLETE MAINTENANCE FOR A STACK
*
* STACK OPERATORS --
*       EMPTY
*       POP
*       PUSH
*       GET
*       INIT
*
* GENERIC PARAMETERS
*     PNAME - NAME OF CAPSULE
*     PSIZE - NUMBER OF ELEMENTS IN THE ARRAY USED AS STACK
*     PTYPE - BASE TYPE OF STACK ELEMENTS
*)

    EXPORTS (EMPTY, POP, PUSH, GET, INIT)#

    (***** GLOBAL CAPSULE VARIABLES *****)
    VAR AT ARRAY[1..PSIZE] OF PTYPE# (* THE STACK *)
        TOP: 0..PSIZE# (* POINTER TO TOP OF STACK *)


    (***** FUNCTION EMPTY *****)

    FUNCTION EMPTY: BOOLEAN#
    (*
    * DETERMINE IF STACK IS EMPTY
    *
    * ARGUMENT DEFINITIONS --
    *     (NONE)
    *)
        BEGIN
            IF TOP = 0 THEN
                EMPTY := TRUE
            ELSE
                EMPTY := FALSE
        END#

    (***** FUNCTION POP *****)

    FUNCTION POP: PTYPE#
    (*
    * POP AN ITEM OFF THE TOP OF STACK
    *
    * ARGUMENT DEFINITIONS --
    *     (NONE)
    *)
        BEGIN
            IF EMPTY THEN
                BEGIN
                    WRITELN(' *** STACK UNDERFLOW IN CAPSULE PNAME ***')#
                    WRITELN('      POP CALLED WITH TOP = 0.')#
                    WRITELN('      EXECUTION TERMINATED.')#
                    HALT
                END
            ELSE
                BEGIN
                    POP := ATOP[TOP]#
                    TOP := TOP - 1
                END
        END#


(***** PROCEDURE PUSH *****)

PROCEDURE PUSH(ITEM: PTYPE)#
(*
* PUSH AN ITEM ONTO THE STACK
*
* ARGUMENT DEFINITIONS --
*     INPUT ARGUMENTS
*       ITEM - DATA ITEM TO BE ENTERED INTO STACK
*)
    BEGIN
        IF TOP = PSIZE THEN
            BEGIN
                WRITELN(' *** STACK OVERFLOW IN CAPSULE PNAME ***')#
                WRITELN('      TOP = STACKSIZE = ',TOP)#
                WRITELN('      EXECUTION TERMINATED.')#
                HALT
            END
        ELSE
            BEGIN
                TOP := TOP + 1#
                ATOP[TOP] := ITEM
            END
    END#

(*$$$$$ PROCEDURE GET *****)

PROCEDURE GET(TOPOFFSET: INTEGER;VAR BOTTOM:BOOLEAN;VAR ITEM:PTYPE)
(*
* READ AN ITEM FROM ANYWHERE ON THE STACK (DOES NOT ALTER STACK)
*
* ARGUMENT DEFINITIONS --
*   INPUT ARGUMENTS
*     TOPOFFSET - INDICATES THE INDEX (RELATIVE TO CURRENT TOP)
*                 OF THE ITEM TO BE READ
*     BOTTOM - INDICATES IF END OF STACK HAS BEEN REACHED
*     ITEM - STACK ENTRY BEING RETURNED
*)
VAR INDEX: INTEGER#  (*INDEX TO ACCESSING STACK ENTRY*)
    BEGIN
        INDEX := TOP - TOPOFFSET#
        IF (INDEX < 1) THEN BOTTOM := TRUE
            ELSE BEGIN BOTTOM := FALSE# ITEM := ATINDEX] END#
    END#


(*$$$$$ PROCEDURE INIT *****)

PROCEDURE INIT#
(* INITIALIZE TOP OF STACK *)
    BEGIN
        TOP := 0#
    END#
END (* CAPSTA CAPSULE *)#
```

PUG

The 6th Annual PACS COMPUTER GAMES FESTIVAL
sponsored by the
Philadelphia Area Computer Society
and
LaSalle College Physics Department
will be held on the 19th of March 1983
from 11:00 A.M. to 4:00 p.m.
in the LaSalle College Ballroom
located at 20th & Olney
Philadelphia, PA 19141

Featuring Computers in Daily Life

For further information contact Stephen A. Longo, Ph.D. Physics Department, LaSalle College, Philadelphia, PA 19141 Phone (215) 951-1255.

Oh! Pascal!

Oh! Pascal! is a book by Doug Cooper and Michael Clancy. Doug Cooper is an excellent programer living in Oakland, California and Michael Clancy directs the introductory programing courses at the University of California Berkley.

Oh! Pascal! has been used at many leading universities as a basic text book for Pascal. These schools include University of California (Berkley), Purdue University, Amherst College, Brandeis University, Harvard and The Rochester Institute of Technology.

The book is very readable and clearly written. It contains an emphasis on general problem solving techniques, an early discussion of procedures, self-check questions and self-test for each chapter. Anti-bugging and debugging sections follow each chapter. There are numerous programing examples of varying difficulty, including long programs. Interactive programs are shown in action, the reader isn't forced to infer their differences from batch. It is 476 pages long and is illustrated. The cost is $15.95 paperback.

Oh! Pascal! is published by W.W. Norton & Company, 500 Fifth Ave., New York, New York 10110.

NEW MODULA-2 VERSION FASTER, EASIER TO USE

DEL MAR, CA, Nov. 30 — Volition Systems has introduced a complete software system based on its fast, easy-to-use version of Modula-2, Niklaus Wirth's powerful new programming language.

"Modula-2 is particularly suited for large industrial and commercial applications. It will save software developers both time and money in program development and maintenance," according to Joel J. McCormack, company president. Volition Systems has pioneered the commercial implementation of Modula-2.

"Our new implementation is faster, more comprehensive, and easier to use then the previous release, which was closely tied to the UCSD Pascal® environment," McCormack said. It can also handle larger programs. The new release, called Version 0.3, conforms to Wirth's recently published book on Modula-2.

Wirth developed Modula-2 (from *MODU*lar *LA*nguage) to replace his earlier language, Pascal. Whereas Pascal was intended as a teaching language, Modula-2 is expressly designed for use in a wide range of real-world applications. The new language — designed to utilize standard software modules — offers great flexibility in the development of large, complex systems.

Volition's Version 0.3 includes a comprehensive module library, a compiler that runs 25 percent faster than the previous version, and a tutorial designed to bring Pascal programmers up to speed on Modula-2 in a matter of hours.

The new version provides all the attractive features of Modula-2: low-level machine access, real-time control, concurrent processes, and type-secure separate compilation with automatic version control. "Interrupt handling is fully supported in Version 0.3 — programmers can now write real-time applications in Modula-2 instead of resorting to error-prone assembly language," McCormack commented.

Version 0.3 is available now for systems based on the 6502 (including the Apple II and ///®), 8080/Z80, TI 9900, and the 68000. Implementations for other popular microprocessors are expected in early 1983, McCormack noted.

The most significant feature in Version 0.3 is the standard library, a collection of modules that offers facilities normally provided by an operating system. The library provides console I/O, random access files, disk directory operations, format conversion, strings, decimal arithmetic, storage management, program execution and process scheduling.

The standard library provides a portable interface to underlying operating systems. Volitions' current Modula-2 system interfaces to UCSD Pascal. Modula-2 implementations for other popular operating systems will be available in 1983.

"With Modula-2, you can develop portable software systems that run without change on a number of different operating systems," McCormack said. "This should be of obvious interst to software developers faced with writing applications which must run on all of today's popular operating systems."

The Modula-2 system also provides access to system-dependent facilities. For instance, Apple users can integrate their existing Pascal and assembly software into the Modula-2 system. And Modula-2 gives them access to the AppleStuff and TurtleGraphics units.

A major goal of the new version was to make the compiler more useful for program development, McCormack said. It can compile larger programs than Volition's previous version and it can compile existing programs 25 percent faster. In addition, the compiler provides conditional compilation facilities and improved error handling.

Modula-2 Version 0.3 is available now from Volition Systems. The complete Modula-2 system includes a fast one-pass compiler, p-code interpreter, module library, the Advanced System Editor (ASE), Pascal compiler, and a complete set of utility programs. The system is proced at $595.

A smaller configuration is available for the Apple II and /// running Apple Pascal. This system includes the Modula-2 compiler, interpreter, and module library. It is priced at $495. Educational, retailer, and distributor discounts are available.

Volition Systems concentrates on systems software development and on research and development in hardware and software. Since the company was founded in 1980, it has been a leader in the implementation and dissemination of the Modula-2 language and other high level languages and in the design and development of advanced computer architectures.

For further information contact: Volition Systems, P.O. Box 1236, Del Mar, CA, (619) 481-2286

TICOM OFFERS THE UCSD P-SYSTEM ON TWO NEW MICROCOMPUTERS

TICOM, developer of integrated office management systems for micro computers, is now the *exclusive* distributor of the UCSD p-System* on the DEC Rainbow 100, and the NEC Advanced Personal Computer (APC). A p-System veteran of four years, TICOM adds these systems to their current list of UCSD p-System based applications packages and development systems for a variety of microcomputers, including the IBM PC and the Xerox 820-11.

TICOM will offer both development and run-time systems as well as its integrated office management software package, FINAL COPY*, on the APC and the Rainbow. FINAL COPY combines word processing, data entry, records processing and remote communications in a single package. This is the same system that has been offered by TICOM on the IBM PC since January 1982.

These products are now availble directly from TICOM. They will be shown on the NEC APC, DEC Rainbow, IBM PC, Xerox 820-11, and the Texas Instruments Business System 200 at COMDEX 82 in Las Vegas, Nov. 29-Dec. 2 in the TICOM, booth #969. Demonstrations on the NEC APC will also be available in the NEC booth, #1734.

TICOM is no newcomer to the p-System. They initially implemented p-System software packages on multi-user minicomputers in 1978. Taking full advantage of the p-System's high degree of transportability, they later adapted it to several different microcomputers.

One additional feature offered on the NEC APC is a graphics implementation. "To completely utilize the APC's extensive graphics hardware capabilities," Michael Hadjioannou, president of TICOM explains, "we have implemented a SIGGRAPH Core compatible set of routines which are callable from UCSD Pascal. Soon to be added will be the ability to access graphics functions with the Presentation Level Protocol (PLP) or Turtlegraphics, making it easy to transport graphics applications to the NEC APC."

Modula 2, Niklaus Wirth's newest programming language, will also be demonstrated on the XEROX 820-11 at the TICOM booth. It, too, is available from TICOM.

For more information stop by booth #969, or contact TICOM at: 13470 Washington Blvd. #207, Marina del Rey, Ca. 90291, (213) 827-7118. Dealer inquiries are welcomed. All press contacts should be directed to Lynn Anderson.

*UCSD p-System is a Trademark of the Regents of the University of California.
*FINAL COPY is a Trademark of TICOM SYSTEMS, Inc.

EDISON AVAILABLE FOR THE IBM PERSONAL COMPUTER

The Edison system is a portable software system for personal computers written by Per Brinch Hansen and described in his book "Programming a Personal Computer" (Prentice-Hall, April 1983).

The Edison system supports the development of programs written in the programming language Edison — a Pascal-like language that supports program modularity and concurrent execution.

The Edison system includes an operating system, an Edison compiler, a screen editor, a text formatter, a print program, and an assembler written in the Edison language.

The program text and portable code of the software are available on diskettes for the following microcomputers:

IBM Personal Computer	PDP 11/23 Computer
32 K words and Keyboard	(or LSI 11) 28 K words
Dual 5¼" Diskette Drive	Dual 8" Diskette Drive
single (or double) sided	RX02 (or RX01)
Monochrome Display	Terminal
Printer	VT 100 (or VT 52)
Display/Printer Adapter	Printer

The software can be edited and recompiled on these machine configurations. It can also be moved to other similar microcomputers by rewriting a kernel of 2 K words.

For more information on the availability of the Edison system and the book, please write to:

Professor Per Brinch Hansen
Computer Science Department
University of Southern California
Los Angeles, California 90089

JRT PASCAL

Since May, when we slashed JRT Pascal's price from $295 to $29.95, we've added over 10,000 new customers! — and we expect to reach 25,000 by year-end!

Needless to say, we're grateful for the deluge of orders. To handle it has taken a new office, new per-

sonnel, and new shipping systems; even then, the mass of orders — a fifty times increase — caused some delays. If your order didn't arrive quickly, thank you also for your patience. We believe you'll find JRT is worth the wait.

With the new capabilities, the goal of a one week order turn-around is now in sight.

Note 1: Five and a quarter inch disk versions

Requiring only 85K of diskette space for the compiler and 35K for the run-time system, JRT is currently the most compact Pascal available for CP/M systems. For program development in JRT Pascal on computers with five inch disk drives, we recommend this file arrangement:

On disk A:
- EXEC,COM
- your editor (ED, Wordstar, etc.)
- the Pascal source program being developed

On disk B:
- JRTPAS2.COM
- PASCAL.LIB
- PASCAL0.INT
- PASCAL1.INT
- PASCAL2.INT
- PASCAL3.INT
- PASCAL4.INT

IMPORTANT NOTE — The file PASCAL.LIB must always be present on the computer system when compiling or executing programs.

Note 2: Patch #1

Applicable version: 2.1
Error: multiplication of real numbers by 0.0 produces incorrect result
Patch procedure: Use CP/M program DDT to patch EXEC.COM — key in underlined code.

```
A>DDT EXEC.COM
DDT VERS 2.2
NEXT PC
5B00 0100
-S563C
563C ED EB
563D 53 ,
-GO
A>SAVE 90
EXEC.COM.
```

Note 3: Patch #2

Applicable version: 2.1
Error: Message 'Source file not found' when compiling under CP/M ver 1.4 or CDOS
Patch procedure: Use CP/M program DDT to patch JRTPAS2.COM — key in underlined code.

```
A>DDT
JRTPAS2.COM
DDT VERS 2.2
NEXT PC
5500 0100
-A2B9
02B9 CALL 3F83
02BC CALL 413D
02BF ,
-GO
A>SAVE 84
JRTPAS2.COM
```

Note 4: JRT Pascal version 2.2 update

Version 2.2 of JRT Pascal is now being shipped — 2.2 includes some internal enhancements and repairs all problems reported in earlier versions. If you want this update, it's yours for the cost of a diskette, postage and handling: $10.

The ONLY disk formats available are:

5¼" for Osborne, Apple CP/M, North Star, Superbrain, Heath hard sector, Heath soft sector, Xerox 820, Televideo

8" single-sided, single density standard

Please specify which of these formats you need.

Note 5: Coming — JRT Pascal version 3.0

In January we'll begin shipping JRT Pascal 3.0 — a major enhancement. New features include:

- builtin indexed file system
- facilities for screen and report formattting
- dynamic arrays
- improved compiler error recovery
- enhanced EXEC interrupt
- full support for file variables and GET/PUT
- expanded user manual

Of course the price of new 3.0 will still be $29.95.

Note 6: Copy and License Policy

We've had lots of questions about our policy on copying JRT Pascal. As our ads say, permission is granted to copy both disk and manual for friends — so long as it's not for resale.

Permission to make copies is also specifically granted to schools and to computer clubs for members.

If you develop application software for resale, you may distribute the run-time system (EXEC.COM and PASCAL.LIB) with your package — with no license or royalty fees.

Note 7: YOUR Pascal application programs

Naturally, more and more owners are developing more and more JRT Pascal written application packages for sale — we've heard from many of them. And — for developers — our copy and license policy is particularly attractive.

Now we're putting together a JRT Application Software Directory and would like to list the packages you have for sale. For free listing, just fill out the enclosed Application Program Description and return it to us with tangible evidence of your package such as brochure, manuals, diskette — but quickly, please: the first Directory is scheduled for February distribution.

Note 8: New address and phone number

The new phone number for orders only is (415) 566-5100.
The address for technical questions and problem reports:

JRT Systems
Technical Services
P.O. Box 22365
San Francisco, CA 94122

The address for new orders:

JRT Systems
550 Irving Street
San Francisco, CA 94122

Note 9: Feedback . . . Please!

A dynamic product, new JRT Pascal versions are always being developed. The system's main evolutionary force is feedback from YOU — the user. We invite — and encourage — you to write us your ideas about how to make JRT Pascal even better.

ENHANCED PASCAL COMPILER FOR IBM MAINFRAME COMPUTERS

ACUMEN Software Services Ltd. is pleased to announce the release of Version 2.0B of the Australian Atomic Energy Commission's PASCAL 8000, an improved Pascal compiler for IBM mainframe computers.

The AAEC's PASCAL 8000 Versin 1.2 was one of the first production compilers for the Pascal language. Version 2.0 offers the user significant improvements; it will run under any of the OS, OS/VS and VM operating systems MFT, MVT, VS1, SVS, VS2, MVS and VM/CMS. It makes full use of the IBM 370's "long" instruction — it has a dynamic dataset allocation — it has improved compilation speed — its modular runtime system makes for easy changes — it enables the user to change the final condition code — it can support lower case. The language accepted by the compiler conforms as closely as possible to the ISO Draft Standard. PASCAL 8000 Version 2.0 can rapidly pinpoint problems in original source language, a function which is available on only a few other compilers.

In Version 2.0B, CMS support for VM/SP has been added, improved traceback in the event of a system abend is provided, compile-time specification of the maximum procedure table size is introduced, as well as other improvements to the run-time system.

PASCAL 8000 compilers are already in successful use in over 250 offices around the world, in banks, schools, life assurance companies, universities, computing firms and government departments. IBM DOS and Perkin-Elmer versions are currently under development.

The compiler is supplied on 9-track EBCDIC 1600 BPI tape and includes; a user reference manual containing a description of the language as implemented, an implementation guide and implementation JCL. PASCAL 8000 has a one-time license charge of $US 2,000 and annual maintenance and enhancements charge of $US 250.

Enquiries about installing a PASCAL 8000 Version 2.0 compiler should be directed to:

Mr. Bryan Brooking
ACUMEN Software Services Ltd.
P.O. Box 86787
North Vancouver, B.C.
V7L 4L3
Telephone (604) 980-7118

USUS FORMS FOUR NEW INTEREST GROUPS, ELECTS OFFICERS AT MEETING IN DALLAS

DALLAS, TX, Nov. 15 — USUS, the UCSD-Pascal User's Society, elected new board members and officers for next year, committed itself to increased user education and informed new special interest groups (SIGs) at the organization's semi-annual national meeting recently concluded here.

Speaking of the strengths of this popular language, keynoter John D. Page of Software Publishing Corp. (Mountain View, CA) noted, "PFS was done in UCSD Pascal because a task of that size and complexity could not be done in BASIC." PFS, with more than 100,000 units sold, is the single best-selling Apple Pascal program.

"As the p-System is becoming more widely distributed and an even more attractive target for application developers, we are experiencing a growing demand for user education," according to Randy Bush of Volition Systems (Del Mar CA), who is the newly elected chairman of the society's board of directors.

"USUS plans to increase its emphasis on tutorials and member education to meet that need," he said. In the future, approximately 40 percent of meeting content will be devoted to tutorials for both users and developers.

Moving in that direction, USUS presented two free-to-the-public tutorials, added four new volumes to its software exchange library and formed four new SIGs at the Dallas meeting. Some 200 people attended it.

SIGs were formed for users of the IBM Personal Computer, Texas Instruments computers and the Sage computers as well as for those interested in influencing file access standards being developed for multi-key access methods for p-System networks.

In addition to Bush, USUS directors for the coming year will be N.C. "Arley" Dealey of Volition Systems, Michael Ikezawa (Rolling Hills, CA), Nancy Lanning of SofTech Microsystems (San Diego, CA) and Robert Peterson of Texas Instruments (Dallas, TX).

Peterson will also serve as president of the organization. Other offiers are A. Winsor Brown (Huntington Beach, CA), vice president; Michael Hadjioannou of Ticom Systems (Marina del Rey, CA), treasurer; and Thomas Woteki of Ferox Microsystems (Arlington, VA).

The IBM PC SIG will have three co-chairs: Gary Gibb of Thunderbird Properties (Oakland, CA), David R. Gobel of Eastern Business Machines (Greenbelt, MD) and Mitchell D. Garrett of Digital Engineering Group, Inc. (Houston, TX).

The TI SIG will be chaired by Danny Cooper (Plano, TX), and Tom Siep of Texas Instruments (Dallas, TX) will head the Sage SIG. Steve Castle (Park Ridge, IL) is chairing the File Access SIG.

In addition to tutorials, SIG meetings and technical sessions, the meeting featured product announcements and hardware demonstrations. SofTech Microsystems announced the availability of its 4.1 version of UCSD Pascal and Statcom (Austin, TX) announced and demonstrated CRTForm, an automatic code generator for UCSD Pascal on 4.0.

Ticom showed the UCSD p-System running for the first time on the NEC Advanced Personal Computer. Other demonstrations included the Sage II computer from Sage Computer Technology (Reno, NV) and the Modula-2 programming language from Volition Systems running on the Sage II, the Apple II and the TI 990.

The next scheduled meeting of USUS is April 22-24, 1983 in San Diego. USUS is a vendor-independent, non-profit user's group for the most widely used, machine-independent software system, UCSD Pascal.

USUS was founded in 1980 to promote and influence the development of the UCSD Pascal System and to provide a forum for education and information exchange about it. Annual membership in the society is $20 for individuals and $500 for institutions.

GREAT PLAINS SOFTWARE ANNOUNCES FIRST SHIPMENT OF THE "HARDISK ACCOUNTING SERIES" TO APPLE DEALERS

Written in USCD Pascal, the program runs on Apple II and III, with a Corvus or profile hardisk. The program will run on IBM's personal computers and most other microcomputers in April.

The menu driven, double entry accounting system features interactive modules and complete audit trials. With extensive data prompts, error checking and an operator's manual, users will find the system easy to use and understand.

For more information contact Great Plains Software, 123 North 15th St., Fargo, ND 58102 or call (701) 293-8483.

INMOS ANNOUNCES OCCAM

INMOS announces occam, a new programming language. Named after the philosopher William of Occam, the language is based on the concepts of concurrency and communication. These concepts enable today's applications to be implemented more effectively and are essential for the complex multi-processor systems of the future.

Systems, even those with only one processor, consist of many parts working together, that is "concurrently." When used in programming a system, occam directly represents these components and their interconnections and gives an efficient design and implementation. Future systems will have many processors, and occam's understanding of concurrency will be essential for their design.

To introduce occam and concurrency, INMOS is offering an Occam Evaluation Kit. This will run on any system supporting the UCSD p-System (version IV), and costs $200. The kit includes a compiler-editor and full supporting documentation. The UCSD p-System may also be purchased with the Occam Evaluation Kit for an additional charge.

Other occam products will become availble in 1983.

INMOS MICROCOMPUTER ACTIVITIES

Inmos is already established as a technical innovator in memory products. It has market leadership in fast 16K static RAMs in both 16K× 1 and 4K×4 organizations. Its IMS2600 64K× 1 dynamic RAM is the fastest available, and it will shortly be introducing 8K×8 and 16K×4 versions.

The other plank in the product strategy is the Transputer, an advanced microcomputer due to be introduced in 1984. It is being designed in Bristol, England at Inmos' United Kingdom Technology Center. Microcomputers are the key products in the semiconductor industry, fuelling the silicon revolution. They are the fastest growing market sector, and with associated hardware and software support products, the largest.

Developing microcomputer systems is a complex task. The user needs efficient tools to design and debug systems and languages to program applications. Inmos decided early that the support products would be made available in the order that the user needed them to create his systems. They will be announced during 1983, ahead of silicon products.

While the transputer will support software in all popular high-level languages available today, it is seen by Inmos as more than just a "better" microprocessor. Rather, it is a silicon "building block", the component for the massively parallel systems of the 80's and beyond, such as the so-called Fifth Generation computer systems.

The efficient design and implementation of these systems is not possible with current languages, whose designers never intended them for such applications. To meet this need occam was created.

WHY A NEW LANGUAGE?

A common factor in real systems is that they consist of a collection of components which exist alongside one another for the lifetime of the system. The components are independent, and from time to time communicate information with one another.

Existing programming languages are designed for single-processor use. Although they do allow a system to be broken down into its separate components, they insist on executing these components sequentially. This is a poor model of a real system.

With the reducing cost and increasing capability of tomorrow's VLSI components, systems can be built from multiple processors, which are much more complex than today's systems. The limitations of current languages prevent the exploitation of such systems, and clearly calls for a new language.

OCCAM

Concurrency in occam is implemented by having a "process" for each independent activity. Concurrency reaches to the lowest level of the language, the individual language statement. These statements are called "primitive processes".

A primitive process on its own cannot do much, so the language provides "constructors" to group them together into bigger processes.

Three types of primitive processes are used in occam. The first and most familiar is the "assignment". Assignment in occam is exactly the same as in other languages; it gives a value to a variable.

The other two primitive processes are "input" and "output". These allow communication between "concurrent processes", that is, processes which are running in parallel. Communication takes place by inputting and outputting "messages" through "channels".

A channel is a one directional link between two concurrent processes. A conversation between two processes requires two channels. A channel implements a handshaken unbuffered data transfer between the sending process and the receiving process. Since a channel is a point-to-point connection, no addresses are needed in the messages.

Occam needs a minimum of constructors. The "sequential" constructor introduces a block of processes which are to be executed one after the other. The "parallel" constructor introduces a block whose component processes are to be executed in parallel.

The "alternative" constructor selects one (and only one) of a set of processes. Each process has a "guard" associated with it which is usually an input statement. The alternative constructor selects the first of its processes whose guard is ready to input and then executes it. If several guards are simultaneously valid, just one of them is randomly selected.

There are also looping and conditional constructors, and a replicator mechanism — which allow the arraying of processes. In addition, the language gives access to a real-time clock.

OCCAM SYNTAX

Occam has been designed to be used with an interactive workstations, which affects aspects of the syntax. For example, since a screen provides a limited number of lines of text, the block structure of the text is shown by indentation (rather than BEGIN..END keywords, which makes inefficient use of the screen). Because the meaning of a program is affected by its physical position on the screen, an integrated editor-compiler is nomally used to write an occam program.

Here are fragments of occam to illustrate the syntax:

```
SEQ                 -- sequential constructor
  in?char           -- first input from channel "in"
  out!char          -- then output the value to channel "out"
PAR                 -- parallel constructor
  out1!'A'          -- output "A" to channel "out1" in parallel
  out2!'B'          -- with outputting "B" to channel "out2"
ALT                 -- alternative constructor
  in1?char          -- guard; try input from channel "in1"
    out!char        -- if guard succeeds, output its input
  in2?char          -- another guard
    out!char        -- and its associated process
WHILE x>0           -- WHILE loop
  SEQ
    in?x            -- input,
    out!x           -- then output as long as x>0
IF x<0              -- conditional
  x:=-x             -- assignment
VAR char:           -- declare a variable, "char"
CHAN in:            -- declare a channel, "in"
VAR array[100]:     -- declare a vector, "array" of 100 elements
CHAN inputs[16]     -- declare a vector of 16 channels
SEQ i=(0 FOR 100) -- FOR loop, sum array elements sequentially
  sum:=sum+array[i]
PAR i=(0 FOR 100) -- "replicator" creates 100 parallel processes
  array[i]:=array[i]+1  -- increments array elements in parallel
ALT i=(0 FOR 100) -- alternative and replicator combined
  inputs[i]?char -- select an input from array of channels
    out!char     -- and output the winner
char:=array(BYTE i) -- BYTE keyword allows byte addressing
PROC buffer (CHAN in,out)  -- abstraction mechanism
  WHILE TRUE               -- loop for ever
    VAR x:
    SEQ                    -- implement a 1-deep buffer
      in?x
      out!x
CHAN c:
  PAR
    buffer (in,c)          -- now invoke the abstraction
    buffer (c, out)
```

OCCAM IMPLEMENTATION

The conventional implementation of a process, which uses an area of memory to hold the variables and scheduling information, works. For many applications, a simple round-robin scheduler is adequate. Many implementations of a channel are feasible and should be readily apparent to system, designers. The details will vary to exploit machine-specific features or other choices, like a multiprocessor implementation. For instance, a channel between processors can use shared memory, IO ports or serial links.

Interrupts are easily handled within occam. A processor with N nestable interrupts can be modelled in occam as N+1 communicating processors. The base processor needs a scheduler, while the interrupt processors may have none; being just a single process waiting for input from a channel which hides the interrupt logic. The microprocessor hardware will then automatically multiplex the processor between base processor and interrupt processors. This ability to handle interrupts in the language can significantly reduce design and integration timescales.

Implementations of occam are efficient, with code densities and execution rates closer to assembler than typical high level languages like Pascal. This is because of a deliberate choice to restrict the language to those features which are supported directly by all likely machines. An implementation of occam needs a small run-time system but this is typically less than 100 machine instructions.

The overheads of concurrency is higher in systems which use "gratuitous concurrency" than in those where the parallelism is tuned for performance. For instance, doing assignment statements in parallel on a single processor system will result in some overhead. However, concurrent communication is efficient and sensible. It is expected that an occam system on an industry-standard microprocessor will incur less overhead than one using a traditional real-time kernel.

WILLIAM OF OCCAM

The language occam was designed by Inmos in conjunction with Professor C.A.R. ('Tony') Hoare, Director of the Programming Research Group at Oxford University.

A predecessor of his at Oxford was the fourteenth century philosopher William of Occam who is best known for "Occam's Razor", "Entia non sunt multiplicanda praeter necessitatem." Literally translated, "entities should not be multiplied beyond necessity", it is often seen as a plea to keep things simple. More generally, it suggests that if two or more solutions to a problem exist, the simplest one is preferred.

This approach of simplicity is fundamental to occam and is extended to all work that Inmos is carrying out in its VLSI products. It also reflects the well-published views of Professor Hoare that many modern languages are unnecessarily complex, and in some cases dangerously so.

OCCAM PRODUCTS

Inmos ia announcing an Occam Evaluation Kit along with the language itself. It allows medium-sized programs to be designed, written and executed, and is intended to teach people to think "parallel".

The kit is a portable compiler and editor built upon Softech's UCSD Pascal system (version IV). It generates p-code, which is executed in the normal fashion by a p-system host. It is available tailored for the Apple 2, Sirius 1/Victor 9000, Intel MDS, IBM Personal Computer, VAX/VMS and LSI/11 and is provided in the appropriate diskette formats for these hosts. It is also available in uncommitted form on 8' diskette in Softech's UCSD Pascal distribution format (single-sided, single-density).

The kit includes language and compiler manuals, together with installation instructions, warranty and example programs. The Occam Evaluation Kit costs $200.

During the first half of 1983, Inmos will announce hardware and software packages which support selected industry standard microprocessors, including the iAPX 86 family and the MC68000 family of microprocessors. These packages will be offered either as "software-only" for running on a UCSD p-system host, or integrated with a microprocessor-based workstation offering high-resolution graphics, 256K bytes of memory and high density floppy-disks. Expansion capability for the workstation will include a local area network and Winchester disks.

For more information on the Occam Evaluation Kit, contact Brad Hartman at INMOS, Colorado Springs, Colorado (303) 630-4362.

TINY PASCAL PLUS+ FOR PET AND APPLE II

ABACUS SOFTWARE announces the release of TINY Pascal PLUS+, an enhanced version of TINY Pascal with support for graphics. The package runs on the 32K PETS and APPLE II's with Applesoft in ROM. It is available for immediate delivery.

TINY Pascal PLUS+ is a complete package allowing the user to create, compile and execute programs written in the Pasal language. TINY Pascal PLUS+ includes:

- LINE EDITOR to create, modify and maintain source
- COMPILER to produce P-code, the assembly language of the P-machine
- INTERPRETER to execute the compiled P-code (with TRACE facility)
- Structured programming constructs: CASE-OF-ELSE, WHILE-DO, IF-THEN-ELSE, REPEAT-UNTIL, FOR-TO/DOWNTO-DO, BEGIN-END, MEM, CONST, VAR, ARRAY

TINY Pascal PLUS+ provides graphics and other built in functions — GRAPHICS, PLOT, POINT, TEXT, INKEY, ABS and SQR. The PET version supports double density plotting on the 40 column screen giving 80 × 50 plot positions. The APPLE II version supports both LORES and HIRES graphics with: COLOR, HGRAPHICS, HCOLOR, HPLOT and PDL. For those users who do not require graphics capabilities, the original TINY Pascal package is still available.

Prices for the diskette versions for APPLE II and PET are $50. A cassette version for the PET is also available for $55. The original non-graphics versions are available for 16K/32K PETS and APPLE II's on diskette for $35 and on cassette for the PEt for $40.

For more information contact: ABACUS Software, P.O. Box 7211, Grand Rapids, Mich. 49510.

HELP WANTED

Our company is presently looking for a Pascal expert to work for us. His duties will include bringing Pascal into the data center as a second language. He/she should have five years experience in Pascal usage, a degree and be a good communicator. This career opportunity is with a major conglomerate and involves state-of-the-art technology.

Please have interested people contact Larry C. McWilliams at 1-800-821-3194.

RIDGE THIRTYTWO GRAPHICS

The RIDGE ThirtyTwo is a 32-bit multi-user graphics work-station. Pascal is the system language.

We are seeking engineering and scientific packages written in Pascal to run on our machine.

The RIDGE ThirtyTwo offers high-performance (2-4 times the speed of a VAX 11/780) and high-resolution graphics (1024×800 pixel graphics displays). I have enclosed results from the Stanford Puzzle Program and the Whetstone Benchmark. Please contact me if you know of any software houses or OEM's who would like to use our high-performance Pascal.

STANFORD PUZZLE BENCHMARK
(Pascal, Subscript version)

Machine (seconds)	Time
IBM-3081	1.3
S-1 Mark I	2.0
IBM-370/168	2.1
Ridge-32	2.2
DEC 2060	5.4
IBM-370/158	7.5
VAX-11/780	10.2
68000 8 Mhz	19.0
IBM 4331	38.0
Apple II	1500.0

The puzzle program, developed at Stanford University by Forest Baskett, tests the computer's ability to perform basic operations, such as procedure calls, array references, conditional branches and comparisons.

WHETSTONE BENCHMARK

Machine	Whetstones (Thousands/sec)
Ridge-32	1500
Perkin Elmer 3240	1172
VAX 11/780	753 (*1168)
Prime 750	750
VAX 11/750	331
DEC 11/34	134
(68000 8 Mhz)	70

The Whetstone program is a floating-point intensive program representative of scientific calculations.

*With addition of hardware floating point.

Please contact Ridge Computers, 586 Weddell Dr., Sunnyvale, California, 94086 or call Benay Dora-Abrams at 408-745-0400.

SOFTWARE CONSULTING SERVICES

901 WHITTIER DRIVE ALLENTOWN, PENNSYLVANIA 18103 (215) 797-9690

PASCAL VALIDATION SUITE VERSION 3.1 NOW AVAILABLE

There are nearly 500 licenses of earlier versions of the Validation Suite. The new Suite is an extensive revision of version 3.0. It contains corrections to nearly 60 deficiencies found in PVS V3.0 and has 553 test programs of which over 150 are new or modified. Subsequent revisions to the Suite are likely to be minor.

The Validation Suite was developed by Brian Wichmann in the U.K. and Arthur Sale in Tasmania under the auspices of the Pascal Users Group. The intention of the package is to encourage a very high degree of portability of Pascal programs (even higher than presently exists), and to provide users with a mechanism to assure themselves that vendor's products comply with the Standard. Validation reports on compilers are published in Pascal News.

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. 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 for the distribution of the results of such use. The Validation Suite has been widely used and distributed, and has not been restricted to a small subset of the user community.

The Way Things Are

The Pascal Compiler Validation Suite consists of approximately 18,000 lines of test code for Pascal compilers. It was developed by A.H.J.Sale and R.Freak of the University of Tasmania and B.A.Wichmann and Z.J.Ciechanowicz of the British National Physical Laboratory. They own it and have the sole rights in determining the policies involved in its distribution. Although the value of the Validation Suite is not directly knowable, one can estimate the cost of recreating it at approximately six dollars per line of code or about $100,000. Drs. Sale and Wichmann have authorized me (as an individual) to act as a distributor for the Validation Suite in both North and South America.

Let Us Help You

1. Should you have any technical questions regarding the Validation Suite, please write to me (don't telephone) and I will respond or forward your commentary to Sale and Wichmann. These men constantly travel and it would be difficult to track them down without my help.

2. If you have trouble reading one of our tapes or diskettes call Martha Cichelli (215-797-9690) and she will help straighten out the problem. Martha is in charge of preparing the distribution.

Please Help Us

If the terms of the license agreement are not acceptable to your organization, please do not request a copy of the Validation Suite. I have neither the right nor the inclination to authorize any amendments to the Sale-Wichmann license agreement.

APPLICATION FOR LICENSE TO USE VALIDATION SUITE FOR PASCAL

Name and address of requestor: _____

(Company name if requestor is a company) _____

_____

Phone Number: _____

Name and address to which information should _____

be addressed (write "as above" if the same): _____

_____

_____

Signature of requestor: _____

Date: _____

In making this application, which should be signed by a responsible person in the case of a company, the requestor agrees that:

a) The Validation Suite is recognized as being the copyrighted, proprietary property of The British Standards Organization and A. H. J. Sale, and
b) The requestor will not distribute or otherwise make available machine-readable copies of the Validation Suite, modified or unmodified, to any third party without written permission of the copyright holders.

In return, the copyright holders grant full permission to use the programs and documentation contained in the Validation Suite for the purpose of compiler validation, acceptance tests, benchmarking, preparation of comparative reports and similar purposes, and to make available the listings of the results of compilation and execution of the programs to third parties in the course of the above activities. In such documents, reference shall be made to the original copyright notice and its source.

Distribution Charge: $300.00
Make checks payable to:
Software Consulting Services
in US dollars drawn on a US bank.
Remittance must accompany application.

Mail Request and Check To:
Software Consulting Services
901 Whittier Dr.
Allentown, PA. 18103 USA
Attn: R. J. Cichelli

SOURCE CODE DELIVERY MEDIUM SPECIFICATION

[ ] Magnetic tape
9-Track, odd parity, 1/2"x600'. Select Density:
[ ] 800 bpi [ ] 1600 bpi
[ ] ANSI-STANDARD. Each logical record is an 80 character card image. Each physical record has a block size of 40 logical records. Select Character Code:
[ ] ASCII [ ] EBCDIC
[ ] Special DEC System Alternate Formats:
[ ] RSX-IAS PIP (requires ANSI MAGtape RSX SYSGEN).
[ ] DOS-RSTS FLX.

[ ] 8" Diskette
[ ] Single Density
[ ] Double Density
Format
[ ] CP/M [ ] UCSD III (W. S. Microengine)
[ ] UCSD II, IV [ ] DEC-RT (Single Density)
[ ] DEC-RSX Files 11 [ ] IBM 3740 (Single Density EBCDIC)
Special Format
[ ] Interleave (1-26)
[ ] Skew (0-25)

Office Use Only

Signed: _____

Date: _____

Richard J. Cichelli

On Behalf of A. H. J. Sale and B. S. I.

0. **DATE** 11/23/82

1. **IMPLEMENTOR/MAINTAINER/DISTRIBUTOR** *(* Give a person, address and phone number. *)*

Foster Schucker
Assistant Director
Computing Services
Suny @ Fresonia
Fredonia, NY 14063
716-673-3393

2. **MACHINE/SYSTEM CONFIGURATION** *(* Any known limits on the configuration or support software required, e.g. operating system. *)*

Burroughs Large Systems
B5000/B6000/B7000
3.2, 3.3, MCP

3. **DISTRIBUTION** *(* Who to ask, how it comes, in what options, and at what price. *)*

Operations Supervisor
Computing Services
Fred Suny @ Fredonia
Fredonia NY 14063
$25 w/tape $50 without (1800)
Ask for 3.3 Pascal

1600 BPI or 800 BPI
Library Maintenance tape

4. **DOCUMENTATION** *(* What is available and where. *)*

Not much but is in computer readable form

5. **MAINTENANCE** *(* Is it unmaintained, fully maintained, etc? *)*

Partially maintained. It's used as a teaching tool, so not much support is really needed

6. **STANDARD** *(* How does it measure up to standard Pascal? Is it a subset? Extended? How. *)*

Have not had a chance to try the sale suite yet. It has extensions to fit into the Burroughs File System. Other minor bells/whistles.

7. **MEASUREMENTS** *(* Of its speed or space *)*

8. **RELIABILITY** *(* Any information about field use or sites installed. *)*

Running at ≅ 25 sites

9. **DEVELOPMENT METHOD** *(* How was it developed and what was it written in? *)*

Step 5 compiler modified by Jim Madden UCSD. Pascal is source Language.

10. **LIBRARY SUPPORT** *(* Any other support for compiler in the form of linkages to other languages, source libraries, etc. *)*

Supports Burroughs intra language library support.

CDC 6000

A version of Pascal 6000 3.2 is now available that uses the ASCII character set (rather than CDC Display Code). If sufficient interest is found, it will be made available for distribution through the standard Pascal 6000 mechanism. Convey your interest to your Pascal 6000 distributor or:

Scott Trappe
MS 92-134
Tektronix, Inc.
PO Box 500
Beaverton, Oregon 97077
(503) 629-1717

CDC 7600 (Manchester)

0. **DATE** 8/15/80

1. **IMPLEMENTOR/MAINTAINER/DISTRIBUTOR** *(* Give a person, address and phone number. *)*

University of Manchester Regional Computer Centre
Oxford Rd., Manchester, England

Maintainer - see 5

2. **MACHINE/SYSTEM CONFIGURATION** *(* Any known limits on the configuration or support software required, e.g. operating system. *)*

Control Data 7600 and CYBER 76
SCOPE 2.1.5, 32/64K/ SCM.

3 **DISTRIBUTION** *(* Who to ask, how it comes, in what options, and at what price. *)*

Contact R. J. Collins at above address. A distribution agreement must be signed and the cost is £50 sterling.

4. **DOCUMENTATION** *(* What is available and where *)*

Same as Pascal 6000 release 3.

5 **MAINTENANCE** *(* Is it unmaintained, fully maintained, etc? *)*

UMRCC cannot undertake to maintain the product although we would be interested in any bugs in the 7600 dependent code.

6 **STANDARD** *(* How does it measure up to standard Pascal? Is it a subset? Extended? How. *)*

Same as 6000 PASCAL release 3.

7 **MEASUREMENTS** *(* Of its speed or space *)*

Requires 50000B words memory to compile most student jobs.

8 **RELIABILITY** *(* Any information about field use or sites installed. *)*

Same as 6000 PASCAL release 3.

9 **DEVELOPMENT METHOD** *(* How was it developed and what was it written in? *)*

Cross compiled from CYBER 7200compiler

10 **LIBRARY SUPPORT** *(* Any other support for compiler in the form of linkages to other languages, source libraries, etc. *)*

Same as 6000 PASCAL release 3.

DEC PDP-11, VAX-11 (Oregon Software)

Oregon Software Pascal-2

0. **DATE** 4 November 1981

1. **IMPLEMENTOR/MAINTAINER/DISTRIBUTOR** *(* Give a person, address and phone number. *)*

Oregon Software
2340 SW Canyon Road
Portland Oregon 97201

Phone: (503)226-7760

2. **MACHINE/SYSTEM CONFIGURATION** *(* Any known limits on the configuration or support software required, e.g. operating system. *)*

All Digital PDP-11 processors, Including Vax-11 in compatibility mode. All Digital PDP-11 operating systems, RSTS/E, RSX-11, RT-11.
Compiler requires EIS, 28K words of memory, 500 blocks of disk.

3. **DISTRIBUTION** *(* Who to ask, how it comes, in what options, and at what price. *)*

Available from above. Write for price and terms.

4. **DOCUMENTATION** *(* What is available and where. *)*

Pascal-2 User Manual, 175 printed pages, includes utility guide. Shipped with order, or write for a free copy.

5. **MAINTENANCE** *(* Is it unmaintained, fully maintained, etc? *)*

Fully maintained.

6. **STANDARD** *(* How does it measure up to standard Pascal? Is it a subset? Extended? How. *)*

Very close to draft standard without conformant arrays.

Extensions include structured constants, "otherwise" in case, I/O interface, Random access I/O, low-level machine interface extensions.

7. **MEASUREMENTS** *(* Of its speed or space. *)*

Code is a small as and as fast as any other Digital Language processor. Benchmark data available on request.

8. **RELIABILITY** *(* Any information about field use or sites installed. *)*

Installed at over 200 sites. Has been used in-house for 2 years.

9. **DEVELOPMENT METHOD** *(* How was it developed and what was it written in? *)*

Written in Pascal, bootstrap using OMSI Pascal-1

10. **LIBRARY SUPPORT** *(* Any other support for compiler in the form of linkages to other languages, source libraries, etc. *)*

Linkage to external routines in Pascal, Macro, of Fortran. Utility programs include cross reference generator, formatter, documentation aids.

Intel 8085 (Cogitronics)

0. **DATE** 28 January 1981

1. **IMPLEMENTOR/MAINTAINER/DISTRIBUTOR** *(* Give a person, address and phone number. *)*

Donald L. Dunstan (503) 645-5043
Cogitronics Corporation
5470 N.W. Innisbrook Pl.
Portland, Oregon 97229

2. **MACHINE/SYSTEM CONFIGURATION** *(* Any known limits on the configuration or support software required, e.g. operating system. *)*

Cogitronics Pascal is configurable to OEM environment.
Target computers: Z-80, 8085
Host computers: GenRad ADS 2300; Tektronix 8002A, 8550; CDC Cyber (6000 series)
Planned host computers: PDP-11; IBM 370; CP/M compatible systems

3. **DISTRIBUTION** *(* Who to ask, how it comes, in what options, and at what price. *)*

Bill Lowery, Director of Marketing
Available on machine readable media of host computers
Single user license $2000
Customer Demonstration Kits available

4. **DOCUMENTATION** *(* What is available and where. *)*

Cogitronics Pascal Reference Manual (available for $15)

5. **MAINTENANCE** *(* Is it unmaintained, fully maintained, etc? *)*

Fully maintained

6. **STANDARD** *(* How does it measure up to standard Pascal? Is it a subset? Extended? How. *)*

ISO standard, see validation suite results
Microprocessor Software Engineering Adaptations

7. **MEASUREMENTS** *(* Of its speed or space. *)*

Z-80 based GenRad development system compiles at 800 source lines per minute
Requires 64K system

8. **RELIABILITY** *(* Any information about field use or sites installed. *)*

Product released 1/1/81

9. **DEVELOPMENT METHOD** *(* How was it developed and what was it written in? *)*

Cogitronics Pascal was written and developed in Cogitronics META compiler generation system.

10. **LIBRARY SUPPORT** *(* Any other support for compiler in the form of linkages to other languages, source libraries, etc. *)*

Linkage is available to externally compiled Pascal modules, externally compiled MICRO language modules, and externally assembled routines.

Intel 8080, 8086 (Microsoft)

0. **DATE** October 28, 1981

1. **IMPLEMENTOR/MAINTAINER/DISTRIBUTOR** *(* Give a person, address and phone number. *)*

Bob Wallace or David Jones
MICROSOFT, INC.
10700 Northup Way
Bellevue, WA 98004

2. **MACHINE/SYSTEM CONFIGURATION** *(* Any known limits on the configuration or support software required, e.g. operating system. *)*

Targets: 8080, 8086, Z8000 under MS-DOS, UNIX, CP/M-80, CP/M-86 and others.
HOST: Above plus DEC-20, VAX, IBM 370 and others.

3. **DISTRIBUTION** *(* Who to ask, how it comes, in what options, and at what price. *)*

Only offered to the Hardware Manufacturers for distribution. Please contact OEM Sales for price and availability.

4. **DOCUMENTATION** *(* What is available and where. *)*

Manual - $20.00.

5. **MAINTENANCE** *(* Is it unmaintained, fully maintained, etc? *)*

Fully Maintained.

6. **STANDARD** *(* How does it measure up to standard Pascal? Is it a subset? Extended? How. *)*

ISO standard (Level 0) plus many extensions for systems programming: strings, address type, super arrays, attributes, value section, interfaces, etc.

7. **MEASUREMENTS** *(* Of its speed or space. *)*

Generates very efficient optimized native code.

8. **RELIABILITY** *(* Any information about field use or sites installed. *)*

Relatively new but well tested.

9. **DEVELOPMENT METHOD** *(* How was it developed and what was it written in? *)*

Developed with DEC-20 Pascal; now self-compiled.

10. **LIBRARY SUPPORT** *(* Any other support for compiler in the form of linkages to other languages, source libraries, etc. *)*

FORTRAN-77 front end available, shared library. Compatible with other Microsoft products.

Intel 8080 (Onacki)

0. **DATE** August 1, 1981

1. **IMPLEMENTOR/MAINTAINER/DISTRIBUTOR** *(* Give a person, address and phone number. *)*

Steve Harrison
Onacki Systems
5161 Cole Street
San Diego CA 92117

2. **MACHINE/SYSTEM CONFIGURATION** *(* Any known limits on the configuration or support software required, e.g. operating system. *)*

Radio Shack's TRS-80 microcomputers, Model I and Model III
Runs under the TRDOS operating system

3. **DISTRIBUTION** *(* Who to ask, how it comes, in what options, and at what price. *)*

Available from Onacki systems
Cost: $239 (discounts available on volume orders, write for information)
Distributed on 5.25 inch diskette
Please specify Model I or Model III microcomputer

4. **DOCUMENTATION** *(* What is available and where. *)*

User manual,, which is included with purchase, describes how to use the compiler and the points of difference with ISO DP185.1 DP7185.1
User manual does NOT contain a tutorial on Pascal

5 **MAINTENANCE** *(* Is it unmaintained, fully maintained, etc? *)*

All questions or comments will be answered by Onacki Systems.

6 **STANDARD** *(* How does it measure up to standard Pascal? Is it a subset? Extended? How. *)*

The principle restrictions from ISO DP7185.1
*Procedural, Functional and Conformant-array parameters are not implemented
*The goto statement is not implemented
*Files have been (slightly) changed to work with TRDOS operateing system

7. **MEASUREMENTS** *(* Of its speed or space *)*

Extremely compact object code format. For example: the compiler is less than 8k bytes

8. **RELIABILITY** *(* Any information about field use or sites installed *)*

An earlier version of this compiler has been in use for 2.5 years

9 **DEVELOPMENT METHOD** *(* How was it developed and what was it written in? *)*
Compiler is written in Pascal and was written and is maintained on a TRS-80 Model I computer with one 5.25 inch disk drive

0 **LIBRARY SUPPORT** *(* Any other support for compiler in the form of linkages to other languages, source libraries, etc. *)*
Additional declared procedures and functions allow access to the TRS-80's graphics, random number generator, etc., as well as access to machine language routines

Intel 8080 (MT Microsystems)

0. **DATE** April 20, 1981

1. **IMPLEMENTOR/MAINTAINER/DISTRIBUTOR** *(* Give a person, address and phone number. *)*

Michael G. Lehman
MT MicroSYSTEMS
1562 Kings Cross Drive
Cardiff, CA 92007
(714) 755-1366

2. **MACHINE/SYSTEM CONFIGURATION** *(* Any known limits on the configuration or support software required, e.g. operating system. *)*

56k 8080/Z80
CP/M required
24 by 80 CRT

3. **DISTRIBUTION** *(* Who to ask, how it comes, in what options, and at what price. *)*

From MT MicroSYSTEMS, on floppy diskettes, $475 (suggested retail)
[no options] includes screen editor w/ program proofreader (checks syntax, spelling, reformats, etc.)

4. **DOCUMENTATION** *(* What is available and where. *)*

185 page User's Guide supplied with system

5. **MAINTENANCE** *(* Is it unmaintained, fully maintained, etc? *)*

Fully maintained by MT MicroSYSTEMS

6. **STANDARD** *(* How does it measure up to standard Pascal? Is it a subset? Extended? How. *)*

ISO Standard with extensions: Dynamic Strings, Modular Compilation
Bit/Byte manipulation, I/O port access, Inline assembly code

7. **MEASUREMENTS** *(* Of its speed or space. *)*

150k bytes of disk space
400 lines/minute on 4 MHz Z80 with 8** floppies

8. **RELIABILITY** *(* Any information about field use or sites installed *)*

More than 1000 field sites installed

9. **DEVELOPMENT METHOD** *(* How was it developed and what was it written in? *)*

Developed from scratch in Pascal, 3-pass recursive descent

10 **LIBRARY SUPPORT** *(* Any other support for compiler in the form of linkages to other languages, source libraries, etc. *)*
Large subroutine library of portable and machine dependent procedures (more than 100 routines)

Mostek 6502 (Abacus)

0. **DATE** January 2, 1981

1. **IMPLEMENTOR/MAINTAINER/DISTRIBUTOR** *(* Give a person, address and phone number. *)*

Abacus Software
P.O. Box 7211
Grand Rapids, Michigan 49510

2. **MACHINE/SYSTEM CONFIGURATION** *(* Any known limits on the configuration or support software required, e.g. operating system. *)*

APPLE II,APPLE II+ with DOS
PET/CBM New ROMS 16K/32K cassette or diskette

3. **DISTRIBUTION** *(* Who to ask, how it comes, in what options, and at what price. *)*

APPLE II/APPLE II+ standard TINY Pascal	$35.	diskette
APPLE II/APPLE II+ graphics TINY Pascal PLUS	$50.	diskette
PET 16K/32K standard TINY Pascal	$40	cassette
PET 16K/32K standard TINY Pascal	$35.	diskette
PET 32K graphics TINY Pascal PLUS+	$55.	cassette
PET 32K graphics TINY Pascal PLUS+	$50.	diskette

4. **DOCUMENTATION** *(* What is available and where. *)*

TINY Pascal User's Manual	$10.	refundable with order of software

5. **MAINTENANCE** *(* Is it unmaintained, fully maintained, etc? *)*

Will correct any problems found by users.

6. **STANDARD** *(* How does it measure up to standard Pascal? Is it a subset? Extended? How. *)*

Subset implementation with graphics extensions for PET and APPLE II.

7. **MEASUREMENTS** *(* Of its speed or space. *)*

8. **RELIABILITY** *(* Any information about field use or sites installed. *)*

Over 200 users of TINY Pascal.
TINY Pascal PLUS+ just released.

9. **DEVELOPMENT METHOD** *(* How was it developed and what was it written in? *)*

BASIC and 6502 Assembly language

10. **LIBRARY SUPPORT** *(* Any other support for compiler in the form of linkages to other languages, source libraries, etc. *)*

Not required

Motorola 6809 (OmegaSoft)

0. **DATE**

1. **IMPLEMENTOR/MAINTAINER/DISTRIBUTOR** *(* Give a person, address and phone number. *)*

OmegaSoft
P. O. Box 70265
Sunnyvale, CA 94086

2. **MACHINE/SYSTEM CONFIGURATION** *(* Any known limits on the configuration or support software required, e.g. operating system. *)*

Motorola 6809 compiler
MDOS version; MDOS09 03.00 8K RAM for operating system plus 24K or more at $2000 for compiler, 2 or more disk drives, at least one drive capable of reading a single-sided disk in the standard MDOS format

FLEX version: 6809 FLEX V3.0, 8K RAM for operating system plus 24K or more at $0 for compiler, 2 or more disk drives, at least one capable of reading an 8 or 5.25 inch single-density, single-sided, soft-sectored disk in the standard FLEX format

Other formats: contact OmegaSoft for availability

3. **DISTRIBUTION** *(* Who to ask, how it comes, in what options, and at what price. *)*

Available from OmegaSoft
Cost $200 with run-time library object
$250 with run-time library and source
Includes compiler, assembler, loader and debugger in object form. utilities in object code and Pascal source, and user manual

4. **DOCUMENTATION** *(* What is available and where. *)*

User manual, included with purchase, available seperately for $20

5. **MAINTENANCE** *(* Is it unmaintained, fully maintained, etc? *)*

6. **STANDARD** *(* How does it measure up to standard Pascal? Is it a subset? Extended? How. *)*

(has HEX, STRING types; only textfiles; origined variables; EXTERNAL procedures; OTHERWISE/ELSE in case statements; no non-local goto's; '***' power operater; string concatenation; and, or, not on numbers)
(May be one of the more complete implemantations of Pascal for micros)

7. **MEASUREMENTS** *(* Of its speed or space. *)*

8. **RELIABILITY** *(* Any information about field use or sites installed. *)*

9. **DEVELOPMENT METHOD** *(* How was it developed and what was it written in? *)*

10 **LIBRARY SUPPORT** *(* Any other support for compiler in the form of linkages to other languages, source libraries, etc. *)*

Additional predeclared procedures and functions for strings, files

Texas Inst. 990 (TI)

0. **DATE** Release 1.7, August 1981

1. **IMPLEMENTOR/MAINTAINER/DISTRIBUTOR** *(* Give a person, address and phone number. *)*

Implemented by Texas Instruments. Information is available from TI sales offices, or write to:
Texas Instruments, Digital Systems Group, MS784, P. O. Box 1444, Houston, Texas 77001 or call (512) 250-7305.

Problems should be reported to: Texas Instruments, Software Sustaining, MS2188, P.O. Box 2909, Austin, Texas 78769 or call (512) 250-7407.

2. **MACHINE/SYSTEM CONFIGURATION** *(* Any known limits on the configuration or support software required, e.g. operating system. *)*

The compiler runs on a TI 990/10 or 990/12 computer under the DX10 or DNOS operating system (TI DS990 system Model 4 or larger, with at least 192K bytes of memory).
The compiled and linked object programs can be executed on any member of the 990 computer family (FS990 or DS990 system) using the TX5, TX990, DX10, or DNOS operating system.

3. **DISTRIBUTION** *(* Who to ask, how it comes, in what options, and at what price. *)*

Available on magnetic tapes, disk pack, or diskettes. Contact a TI salesman for a price quotation and further details.

4. **DOCUMENTATION** *(* What is available and where. *)*

The TI pascal language is specified in the TI Pascal Reference Manual TI part number 2270519-9701. Instructions for using the compiler and linking and executing Pascal programs are given in the ''DX10 TI pascal Programmers Guide'', part number 2270528-9701 and the ''DNOS TI Pascal Programmers Guide'', part number 2270517-9701.

5. **MAINTENANCE** *(* Is it unmaintained, fully maintained, etc? *)*

TI Pascal is a full supported product. Bug reports are welcomed and maintenance and further developmentswork are in progress.

6. **STANDARD** *(* How does it measure up to standard Pascal? Is it a subset? Extended? How. *)*

TI Pascal has some differences from standard Pascal. The major differences are:
* A goto cannot be used to jump out of a procedure
* The control variable of a FOR statement is local to the loop.
* The precedance of Boolean operators has been modified to be the same as in Algol and FORTRAN
* The standard procedures GET and PUT have been replaced by generalized READ and WRITE procedures.

TI Pascal has many extensions to standard Pascal including random access files, dynamic arrays, ESCAPE and ASSERT statements, optional OTHERWISE clause on CASE statements, and formatted READ.

7. **MEASUREMENTS** *(* Of its speed or space. *)*

The compiler occupies a 64K byte memory region.

8. **RELIABILITY** *(* Any information about field use or sites installed. *)*

The system has been used by several different groups within TI since October of 1977, and by a number of outside customers since May of 1978. Updates have been released in January 1979, January 1980 and August 1981. This long history of extensive use and maintenance make this a stable and reliable product.

9. **DEVELOPMENT METHOD** *(* How was it developed and what was it written in? *)*

The compiler produces object code which is link-edited with run-time support routines to form a directly executable program. The compiler is written in Pascal and is self-compiling.

10. **LIBRARY SUPPORT** *(* Any other support for compiler in the form of linkages to other languages, source libraries, etc. *)*

TI Pascal supports seperate compilation of routines and allows linking with routines written in FORTRAN or assembly language.

Zilog Z-80 (Ithaca)

0. **DATE** May 12, 1981

1. **IMPLEMENTOR/MAINTAINER/DISTRIBUTOR** *(* Give a person, address and phone number. *)*

Ithaca Intersystems, Inc. PASCAL/Z
1650 Hanshaw Road
P.O. Box 91
Ithaca, New York 14850

2. **MACHINE/SYSTEM CONFIGURATION** *(* Any known limits on the configuration or support software required, e.g. operating system. *)*

Z-80 system with minimum of 56K memory (including the CP/M operating System) and one disk drive.

(64K and two disk drives recommended for serious program development)

3. **DISTRIBUTION** *(* Who to ask, how it comes, in what options, and at what price. *)*

Package includes object code for Pascal/Z compiler, ASMBLE/Z macro-assembler, LINK/Z linker/loader and SWAT, our interactive symbolic debugger. The libraries are provided in both object and commented source code (Z-80). Also included are a number of support and example files and programs, and documentation. Available on CP/M compatible 8" diskettes from the distributor--contact Intersystems for information on obtaining other formats.
U.S.(Dom)retail price: $395.00

4. **DOCUMENTATION** *(* What is available and where. *)*

Over 300 pages of documentation, including: the Pascal/Z Implementation Manual, ASMBLE/Z , LINK/Z and SWAT manuals and the Jensen & Wirth USER MANUAL AND REPORT.

5. **MAINTENANCE** *(* Is it unmaintained, fully maintained, etc? *)*

Updates approximately every three months, available for a nominal charge to registered users.

6. **STANDARD** *(* How does it measure up to standard Pascal? Is it a subset? Extended? How. *)*

Closely follows Jensen & Wirth definition. Exceptions: No GET/PUT (our READ/WRITE routines have been expanded to handle all I/O functions), no PAGE, no procedural parameters. Extensions: Direct File Access, Variable length strings, EXTERNAL routines, separate compilation, INCLUDE files, variant records implemented, ELSE on the CASE statement.

7. **MEASUREMENTS** *(* Of its speed or space. *)*

Running the Erasthones sieve on page 54 of the USER MANUAL AND REPORT (with a WRITE statement added to display) 6K COM file ran in 45.85 seconds, much better than competition.

8. **RELIABILITY** *(* Any information about field use or sites installed. *)*

Over 1500 Users. No bugs found by users in current release (3.3), which has been released for three months.

9. **DEVELOPMENT METHOD** *(* How was it developed and what was it written in? *)*

Developed in Pascal.

10. **LIBRARY SUPPORT** *(* Any other support for compiler in the form of linkages to other languages, source libraries, etc. *)*

Libraries included in source. Alternate libraries available from Z Users' Group. Assembler outputs Microsoft-compatible REL files -- can be linked to other languages using our LINK/Z, provided protocol is same.

Zilog Z-80 (Cogitronics)

See Intel 8085.

Zilog Z8000 (Microsoft)

See Intel 8080, 8086.

Zilog Z-80 (MT Microsystems)

See Intel 8080.

Machine Index

Machine (operating system)	Issue:page	Comments
ALL	#15:101	Pascal I (Derived from Pascal S)
BESM-6	#15:107	
Burroughs B5700	#15:107	
Burroughs B6700/B7700 (MCP)	#19:113	
CDC 6000	#19:115	
CDC 6000	#15:106	
Cyber 70 and 170	#15:106	
DEC PDP-11	#15:111	
DEC PDP-11	#15:112	UCSD Pascal
DEC PDP-11	#19:115	UCSD Pascal
DEC PDP-11	#15:124	
DEC PDP-11 (RSTS)	#15:100	Pascal S
DEC PDP-11 (RSX-11M/IAS)	#17:86	
DEC PDP-11 (RSX-11M/RT-11)	#15:101	Concurrent Pascal
DEC PDP-11 (Unix)	#15:111	
DEC PDP-11 (Unix)	#15:100	Pascal E
DEC PDP-11 (Unix)	#15:103	Modula
DEC PDP-15	#15:124	
DEC VAX	#17:89	
DEC VAX (Unix)	#19:115	
DG Eclipse	#17:106	
DG Eclipse (AOS)	#15:110	
DG Eclipse (AOS)	#15:109	
DG Eclipse (RDOS)	#15:106	
DG Nova (AOS)	#15:110	
Digico Micro 16E	#15:113	
Facom 230-45S	#15:112	
General Electric GEC4082	#15:113	
Golem B (GOBOS)	#17:104	
HP 1000	#19:116	
Honeywell 6000/Series 60 Level 66 (GCOS III)	#15:113	
Honeywell Level 6	#15:113	
IBM 3033	#19:120	
IBM 360/370	#15:115	
IBM 360/370	#15:114	
IBM 370	#15:124	
IBM 370	#17:104	
IBM 370	#17:102	
IBM 370	#19:117	
IBM 370/303x/43xx	#19:117	
IBM Series 1	#19:116	
IBM Series 1	#15:114	
ICL 1900	#15:116	
Intel 8080/8085	#15:119	
Intel 8080/8085	#15:118	
Intel 8080/8085	#15:119	
Intel 8080/8085	#15:117	
Intel 8080/8085	#17:102	
Intel 8080/8085 (CP/M)	#17:105	
Intel 8080/8085 (TRS-80/Northstar)	#15:100	Tiny Pascal
Intel 8086	#15:119	
Intel 8086	#15:108	Modula
MOS Tech 6502 (Apple)	#15:107	UCSD Pascal
Modcomp II and IV	#15:120	
Motorola 6800	#17:102	
Motorola 6800	#15:120	
Motorola 6800	#19:121	
Motorola 6800	#19:120	
Motorola 6800 (Flex)	#15:123	
Motorola 68000	#19:121	
Motorola 6809	#15:103	Modula
Motorola 6809 (MDOS09)	#17:102	
Nord 10 and 100 (Sintran III)	#15:121	
Perkin-Elmer 3220	#15:122	
Perkin-Elmer 7/16	#15:121	
RCA 1802	#17:103	
RCA 1802	#15:122	
Siemens 7.748	#15:124	
Sperry-Univac V77	#15:124	
Texas Instruments 990	#17:101	
Texas Instruments 9900	#15:124	
Zilog Z-80	#15:124	
Zilog Z-80	#17:86	
Zilog Z-80	#15:124	

Machine (operating system)	Issue:page	Comments
Zilog Z-80	#17:104	
Zilog Z-80	#19:123	
Zilog Z-80 (CP/M0)	#17:103	
Zilog Z-80 (TRS-80)	#15:124	
Zilog Z-80 (TRS-80)	#19:124	
Zilog Z80	#15:118	
Zilog Z80	#15:119	
Zilog Z8000	#15:119	

HP 3000 Series 33

Authors: Paul J. Campbell and Charles R. Williams
Beloit College, Beloit, WI 53511, USA

Pascal Processor Identification

Computer: Hewlett-Packard 3000 Series 33 running under operating system HP 32033 MPE IV Version C.DO.20

Processor: Pascal/3000 Version HP 32106A.00.03, which Hewlett-Packard asserts is an extension of the proposed ANSI Standard Pascal (May 20, 1981 version).

Installation: Beloit College, Beloit, WI 53511, USA

Test Conditions

Tested by: Paul J. Campbell and Charles R. Williams
Date: July-August 1982
Validation Suite Version: 3.0, (issued 8 January 1982), which appears to test agreement with DP7185.1, the second draft of the proposed ISO Pascal Standard

Report Sent To:

Lance Carnes, Editor, HP-3000 Special Interest Group for Pascal (SIGPascal), TEXET Company, 163 Linden Lane, Mill Valley, CA 94941
William J. Cody, Applied Mathematics Division, Argonne National Laboratory, Argonne, IL
Jean Danver, Hewlett-Packard Company, Information Systems Division, 19420 Homestead Rd., Cupertino, CA 95014
Lloyd D. Davis, Editor, Newsletter, HP-3000 Special Interest Group for Education (SIGED), Director, Academic Computing Services, 209 Hunter, The University of Tennessee at Chattanooga, Chattanooga, TN 37402
Bob Dietrich, M.S. 92-134 Tektronix, Inc., P.O. Box 500, Beaverton, OR 97077
Charley Gaffney, Pascal News, 2903 Huntington Road, Cleveland, OH 44120
William M. Kahan, Computer Science Dept., University of California, Berkeley, Ca 94720
Emil Knorr, Math. Dept., Shaker Heights High School, Shaker Heights, OH
John Nierengarten and Dan Abts, Computer Center, University of Wisconsin, LaCrosse, WI
John R. Ray, Editor, Journal of the HP-3000 International Users Group, The University of Tennessee-Knoxville, Knoxville, TN 37401
Mike Riedel, Software Engineer, Hewlett-Packard, 150 S. Sunny Slope Rd., Brookfield, WI 50005
Arthur Sale, Dept. of Information Science, University of Tasmania, Tasmania, Australia
Richard Sours, Math. Dept., Wilkes College, Wilkes-Barre, PA
B. A. Wichmann, NPL

Introduction

"Pascal/3000 [also referred to as HP3000 Pascal] is a superset of Hewlett-Packard Standard Pascal . . .; HP Standard Pascal, in turn is a superset of American National Standards Institute (ANSI) Pascal." (Pascal/3000 Reference Manual, p. 1-1).

Programs in the validation suite were compiled with the compiler option ANSI ON, so that the compiler would issue a warning when it encountered features not legal in ANSI Standard Pascal. In the sections below, warnings of this nature are either mentioned explicitly or the feature involved is marked as a feature of HP Standard Pascal. The validation suite itself contains some defective tests. Those previously reported by Wichmann[1983] are marked "ignore test output per Wichmann."

Principal Deviations

- GET is implemented as a "deferred" get in order to facilitate interactive I/O
- real numbers are not written correctly to files
- a FOR loop variable may be altered from within its loop, and it is still defined after completion of the loop
- pointers which are still needed are allowed to be disposed, and pointers with explicit tag values are handled incorrectly
- a procedure call may be bound to a wrong defining occurrence
- the LN function has large relative errors (about 10%) for arguments near 1

Main Extensions

- OTHERWISE and subrange-like lists may be used as case-selector elements
- the predefined type LONGREAL is available
- the predefined type STRING is implemented as a PACKED ARRAY OF CHAR with a declared maximum length and an actual length that may vary at runtime
- primitives are provided for manipulation of objects of type STRING
- a function may return an object of structured type
- constructors are available for assigning constant values to objects of structured types
- values of user-defined enumerated types can be directly written to and read from files
- a packed array of CHAR can be read with a single READ command
- a subprogram in any of the languages SPL, Fortran/3000, Cobol/3000, and Pascal/3000 can be called by a program in any of the other languages
- conformant arrays are not handled

CONFORMANCE TESTS

Number of tests	run	154
	invalid	3
	irrelevant	0
	passed	149
	failed (number of causes)	2 (2)
	detecting compiler bugs	0

Tests invalid

6.4.3.3-5 — Ignore test output per Wichmann. Compiler does not permit an uniitialized empty record to be accessed.

6.5.1-1 — Ignore test output per Wichmann. Section 6.10 of 7185.1 (ISO second draft) demanded that "Each program parameter shall be declared in the variable-declaration-part of the program-block." The wording of 6.10 was changed in the ISO third draft to "Each program parameter shall have a defining-point as a variable-identifer for the region that is the program block." Either wording affects the parameters "name" and "firstname" in this program.

6.6.6.5-1 — Ignore test output per Wichmann. Compiler issues no warning or error message.

6.9.3.5.1-1 — Ignore test output per Wichmann. Still, the floating-point representation of real numbers is not written correctly to textfiles. The compiler fails to write the initial space required before each non-negative number. (Note: The test does not check writing of negative reals.)

Details of failed tests

6.7.1-2 — Compiler rejects [x. .y] where x > y, claiming that a "set of this size cannot be constructed." The standard requires the expression to be interpreted as the empty set.

6.9.3.5.2-1 — The fixed-point representation of real numbers is not written correctly to textfiles, as the number 0.0 is written a .0 instead of the required 0.0. The compiler omits the initial zero for all positive reals between 0 and 1.

EXTENSION TESTS

Number of tests run 3

Details of tests

6.1.9-7 — Equivalent relational symbols are not defined.

6.1.9-8 — None of the alternate symbols %, %=, .= is defined.

6.8.3.5-16 — The alternative OTHERWISE is accepted in a CASE statement (HP Standard Pascal feature).

DEVIANCE TESTS

Number of tests		
	run	115
	invalid	2
	irrelevant	1
	correctly detecting deviations	91
	true extensions	13
	not detecting deviations and not true extensions (number of causes)	8 (4)
	detecting compiler bugs	0

Tests invalid

6.3-7 — The syntax in lines 14 and 15 is incorrect: the caret symbol ^ should be deleted in both. With these corrections the compiler deviates from the standard by allowing the use of NIL in the CONST section.

6.4.6-6 — Program fails to declare program parameter f within the VAR section, as required by the standard. With f correctly declared, the compiler passes the test.

Tests irrelevant

6.4.3.3-7 — This test relies on the compiler deviating for tests 6.4.3.3-10 through 6.4.3.3-13, which it does not.

True extensions

6.1.7-9 — Compiler permits assignment of a single character, quoted or unquoted, to a PACKED ARRAY OF CHAR of any positive size. (HP Standard Pascal feature). It does not allow assignment to a variable of type CHAR of a (padded or not) PACKED ARRAY OF CHAR containing a single character.

6.1.7-10 — Padding with spaces is done automatically in assigning a shorter string to a longer one. (HP Standard Pascal feature).

6.1.7-11-11 — Assignment of a null string is permitted. (HP Standard Pascal feature)

6.1.7-12 — String constants are indexed.

6.1.8-5 — Space may be omitted between a number and a following word-symbol.

6.2.1-8,6.2.1-10 — Multiple declaration parts are allowed: the CONST, TYPE, and VAR sections can be repeated and intermixed. The LABEL section must still precede, and the procedure and function sections follow, the block of CONST, TYPE, and VAR sections. (HP Standard Pascal feature)

6.3-9 — The value of a declared constant may be specified with a constant expression. (HP Standard Pascal feature)

6.4.3.3-8 — A warning instead of an error message is issued if a case label is not within tag or select expression range.

6.4.5-12 — To compare two string literals, the compiler blank-fills a shorter one. (HP Standard Pascal feature)

6.6.1-5 — Formal parameters may be repeated in the subsequent procedure declaration of a FORWARD procedure. (HP Standard Pascal feature)

6.6.2-5 — A function may return a set, an array, or a record instead of an object of simple type. (HP Standard Pascal feature)

6.8.3.5-7 — Subrange-like lists may be used as case-selector elements. (HP Standard Pascal feature)

Deviations not detected

6.2.1-6 — Declared but unused labels are allowed. (Note: Such behavior was not prohibited in the first draft of the ISO standard, but is prohibited in the second and third drafts at 6.2.1.)

6.6.1-3, 6.6.1-4 — A procedure call may be bound to a wrong defining occurrence: in these cases, to the outer of the two wrong procedures.

6.6.3-4 — A variable parameter is allowed to denote a field which is the selector of a variant-part.

6.8.3.9-7 — Assignment may be made within the loop to a FOR loop control variable.

6.8.3.9-8 — Compiler fails to detect use of a FOR loop control variable after completion of the loop. The value of the variable after completion is the final-value in the FOR statement.

6.8.3.9-9 — After a FOR loop which is not entered, the value of the control variable is defined but unknown.

6.8.3.9-16 — The control variable of a FOR loop may be reassigned by a READ during execution of the loop.

ERROR-HANDLING

Number of tests		
	run	55
	invalid	2
	irrelevant	0
	passed	34
	failing to detect errors (number of causes)	18 (8)
	detecting compiler bugs	1

Tests invalid

6.6.6.5-6 — The test considers it an error if after REWRITE(fyle), EOF(fyle) is defined. In fact the standard requires EOF(fyle) to be true under this circumstance; it is fyle^ that is required to be undefined. The compiler abides by the standard.

6.9.3.2-3 — The statement REWRITE(f) must be inserted before the call to write to f. With this correction, the compiler passes the test.

Details of tests failing to detect errors

6.4.3.3-10 through 6.4.3.3-13 — Undefined tagfields in variant records are not detected.

6.5.5-2 — Compiler fails to detect the change in value of a file buffer variable when used as a global variable while the buffer variable's dereferenced value is passed as a VAR parameter.

6.5.5.3 — As for 6.5.5-2, except that here the buffer variable is an element of the record variable list of a WITH statement.

6.6.2-9 — Compiler does not detect that a function identifier has not been assigned a value within the function; the standard requires such a function identifier to be undefined. (The test would be enhanced by revealing what value (if any) is assigned by execution of the function.)

6.6.5.3-6 — Compiler fails to detect disposing of a pointer variable which refers to a current actual VAR parameter.

6.6.5.3-7 — Compiler fails to detect disposing, within the scope of a WITH statement, of a pointer variable which refers to an element of the current record-variable-list of the WITH.

6.6.5.3-8 through 6.6.5.3-10 — Compiler fails to detect errors in the use of a pointer variable that was allocated with an explicit tag value.

6.6.5.3-11 — Pointer still usable after DISPOSE.

6.6.6.5-7 — Compiler fails to detect error of applying EOLN function to a file for which EOF is true.

6.7.2.2-13 — Error of a negative right operand in MOD is undetected. (The test would be more valuable if it revealed how a compiler accepting this construct handles it.)

6.8.3.9-19 — A FOR statement control variable is still defined after the loop is completed; its value is the final-value in the FOR statement.

6.9.3.2-4, 6.9.3.2-5 — Compiler detects no error when asked to write a real number using 0 digits after the decimal point.

Tests detecting compiler bugs

6.6.5.2-5 — In order to facilitate I/O with interactive devices, GET is deliberately implemented as a "deferred" GET, which postpones the actual loading of a component into the buffer variable. Also deferred are setting the file buffer to undefined and EOF to true. Hence the compiler should not conform to the standard's pre- or post-assertions for GET. However, runs of 6.6.5.2-5 at two different times produced inconsistent results, the compiler failing the test on one occasion and passing it the other.

IMPLEMENTATION-DEFINED

Number of tests		
	run	14
	invalid	2
	irrelevant	0
	detecting compiler bugs	0

Details of invalid tests

6.6.6.1-1 — A standard function may not be used as parameter to a procedure.

6.6.6.2-11 — Because this test relies on non-detection of underflow at runtime, the procedure MACHAR has to be modified to trap run-time underflow and continue execution. (This is accomplished by using the compiler library routine XARITRAP). Even with this modification, the program fails to produce results completely agreeing with known features of the processor.

VARIABLE	MEANING	PROGRAM VALUE	TRUE VALUE (where different)
beta	radix	2	
t	number of digits in floating point significand	23	
rnd	rounds	1	
ngrd	number of guard digits for multiplication	0	
maxexp	$1.0 - 2^{maxexp} <> 1.0$	28	
minexp	$1.0 - 2^{minexp} <> 1.0$	22	
iexp	number of bits (including sign) reserved for exponent	9	
minexp	minexp is smallest floating point power of 2	95	
maxexp	maxexp is largest floating point power of 2	257	255
eps	$1.0 + eps <> 1.0$	1.192093 E-07	
epsneg	$1.0 - epsneg <> 1.0$	2.384186 E-07	
smin	smallest floating pt. power of 2	1.727254 E-77	
xmax	largest floating point number	1.727253 E-77	1.15792 E+77

The program assumes that maxexp can be calculated by adding minexp to a power of 2. This reasoning fails to account for computers like the one at *hand*,

which have a single exception to their assumption of a leading 1 preceding the mantissa of a floating point number: namely, the number with exponent zero and mantissa zero is interpreted as 0.0, instead of as 2^{-256}. In fact the compiler can represent all floating point numbers (within its range of precision) between 2^{-256} and 2^{256}, not including these lower and upper bounds. The smallest floating point number is

(1 + eps) 2^{-256} and the largest is (1 − epsneg) 2^{256}

In the other tests using MACHAR, the procedure is replaced by one simply assigning the known correct values.

(The following changes should be made in the text of the long initial comment of the test 6.9.2-6 should be 6.9.1-6 negeps should be negep, and it is the largest *in magnitude* negative integer . . .)

Details of implementation-dependencies

6.1.9-5 — The alternate comment delimiters (*, *) are implemented.

6.1.9-6 — The equivalent symbols @ for up-arrow and (. , .) for braces are implemented.

6.4.2.2-10 — MAXINT = 2147483647 = $2^{31}-1$

6.4.3.4-5 — The base-type of a set may have as many as 32768 elements, according to Pascal/3000 Reference Manual.

6.7.2.3-3, 6.7.2.3-4 — In test of short-circuit evaluation of (A AND B) and (A OR B), only the first expression A is evaluated. It is possible to force full evaluation by using the compiler command PARTIAL_EVAL OFF, the default being ON.

6.8.2.2-1, 6.8.2.2-2 — Evaluation precedes selection in the assignments A[I] := expression, p^ := expression.

6.8.2.3-2 — Actual parameters to a procedure are evaluated in forward order.

6.9.3.2-6 — default field widths are

integer	:	12 characters
boolean	:	varies according to the boolean value
real	:	12 characters

6.9.3.4.1-2 — The number of digit characters written in the exponent of a real value expressed in floating-point format is 2.

6.9.3.6-1 — The representations of true and false, with parentheses to indicate width, are (TRUE) (FALSE)

QUALITY

Number of tests	run	61
	invalid	0
	irrelevant	0
	passed	48
	failed (number of causes)	13 (9)
	detecting compiler bugs	0

Details of some tests passed

1.2-1 — General check on execution speed: the program executes in 11.2 sec., corresponding to 89 thousand whetstone instructions per second.

1.2-2 — GAMM measure: The program executes 3 million GAMM units in 160.8 sec, for a GAMM measure of 53. The values printed are ACC = 16.7319145, ACC1 = .0016733; the value for ACC should be 16.73343.

1.2-3 — Speed of procedure calls: The program contains 228,057 procedure calls, and executes in 20.0 sec., for an average of 11,400 calls per second, or an overhead of 88 microseconds per call.

6.4.3.4-4 — Warshall's algorithm executes in 0.8252 sec. (average of five runs) and requires 2330 bytes of storage for all varibles.

6.6.5.3-12 — This test program must be compiled with the Pascal/3000 compiler option HEAP_DISPOSE ON; the default setting is OFF. (This option is not available in the HP Standard Pascal subset.)

6.8.3.5-12 — Use of a case constant of the same base type as the case selector — but outside the subrange of the case selector type — results in a compile-time error.

Details of tests failed

6.1.5-9 — Very large values: Each very large value produces an error message.

6.1.8-6 — Compiler fails to issue a warning for a possible unclosed comment.

6.4.3.2-6 — The index type of an array may not be INTEGER, and the compiler prints an appropriate error message.

6.4.1-2 — Fewer than 300 identifiers are allowed in a declaration list.

6.6.6.2-8 — Test of EXP function produces loss of 7 base 2 significant digits for arguments −103.762 and 115.1674. See note below on 6.6.6.2-10.

6.6.6.2-9 — Tests of SIN and COS functions produce respective losses of 16 and 15 base 2 significant digits for respective arguments 18.84967 and 23.56232. See note below on 6.6.6.2-10.

6.6.6.2-10 — Test of LN function fails because of large relative errors (about 10%) for arguments near 1. Since the Pascal/3000 compiler calls system library routines to calculate EXP, SIN, COS, and LN, other compilers and interpreters which also use those routines (e.g., Fortran/3000, Basic/3000, etc.) inherit the same inaccuracies.

6.8.3.4-2 — IF statements can be nested only 11 deep, not 24.

6.8.3.5-15 — CASE statements can be nested only 11 deep, not 15.

6.8.3.8-3 — WHILE statements can be nested only 14 deep, not 15.

6.8.3.9-20 — FOR loops can be nested only 11 deep, not 20.

6.9.1-8 — Test of accuracy of read/write for reals fails. Result was too large 47 times, equal 0 times, and too small 53 times. See 6.9.3.5.2-2 for underlying explanation.

6.9.3.5.2-2 — Test to check accuracy of write for reals produces repeated error message "input incorrect — nondigit read." The standard (6.9.4.5.2 of second draft, 6.9.3.4.2 of third draft) requires that WRITELN (X:33:30) write 30 digits after the decimal point. Pascal/3000 Reference Manual (p. 6-41) notes that in no case will more digits be printed than are in the internal representation. The input errors ("non-digit read") are from all of the leading blanks the compiler inserts to right-justify the shorter output. Using just WRITELN(X) gives agreeable results. (The behavior of this compiler seems more reasonable than that prescribed by the standard.)

LEVEL 1 (CONFORMANT ARRAYS) TESTS

Number of tests	run	11
	irrelevant	11

6.6.3.7-1 through 6.6.3.7-10, 6.6.3.8-1 — Ignore test output per Wichmann. Conformant arrays are not handled by Pascal/3000.

Concluding Comments

Compiler errors discovered by users of Pascal/3000 and reported to Hewlett-Packard are published monthly in the Software Status Bulletin for Program Team 3000. Most of these errors involve extension or other feature which do not involve the Pascal standard, but some involving the standard were not caught by the Validation Suite:

- Integer multiplication by (−1) crashed an earlier version (Version 00.00) of the compiler
- The invalid use of declared variables which are accessed within binary and unary expressions — but which never have values assigned to them — is not always detected, although one instance was caught by Test 6.2.1-11
- compiler erroneously allows redefinition of the reserved word WRITE as the name of a procedure

(The Software Status Bulletin also features sometimes-amusing advice under "Temporary Solution," such as

- Ignore it [the message to inform HP if a certain error occurs], your program is correct and can be run as is.
- Use a real file name [instead of ']
- Do not take advantage of the fact that this error is not detected, because it will be.)

References

Addyman, A., *et al.*, ISO DP/7185 — A Draft Proposed Standard for the Programming Language Pascal, *Pascal News* Number 18 (May 1980) 2-70. ["ISO First Draft]

Differences Between the Draft International and American Pascal Standards, X3J9/82-102 JPC/82-102, 5 pp.

DP7185 Specification for the Computer Programming Language Pascal 97/SC 5 N 595 (January 1981), *Pascal News* Number 20 (December 1980) 1-83. ["ISO Second Draft"]

DP7185 Specification for the Computer Programming Language Pascal 97/SC 5 N 6d78 (4 November 1981), 88 pp. ["ISO Third Draf"]

HP 3000 Support Systems, Pascal/3000 Reference Manual, 1st Edition, December 1981.

Joint ANSI/X3J9 IEEE Pascal Standards Committee, American National Standard Programming Language Pascal, Second Draft, 15 July 1982, Foreword + 81 pp.

Software Status Bulletin for Program Team 3000.

Wichmann, B.A., Status Report on Version 3.0 of the Pascal Test Suite, *Pascal News* Number 24 (January 1983) 20-22. **PUG**

Intel 8085, Zilog 80 (Cogitronics)

Pascal Processor Identification

Target computers: Z80, 8085
Host computers: GenRad ADS 2300; Tektronix 8002A, 8550; CDC Cyber (6000 series)
Planned host computers: DEC PDP-11; IBM 370; CPM compatible systems
Processor: Cogitronics Pascal V1.2C

Test Conditions

Time: December 1980
Tests carried out by: D. Dunstan
Validation Suite Version: 2.2

Restrictions and Extensions

Due to the byte addressable nature of the target machines, PACK and UNPACK procedures are not supported.

PACK is ignored in declarations.

Strings are compatible if their lengths are the same. The lower bound of the index type need not be one.

No runtime checks are made.

The result of a function may be any data type (other than file.)

Procedures and functions may not be used as parameters.

PAGE procedure is not supported.

A GOTO target must be within the current routine or the mainline.

No restrictions are placed upon the FOR loop control variable.

The standard files INPUT and OUTPUT are always opened automatically whether or not they are mentioned on the program header.

Implicit references to the standard files INPUT and OUTPUT are always possible, even when the identifiers INPUT and OUTPUT have been redefined.

Conformance Tests

Number of tests attempted: 139
Passed: 127
Failed due to restrictions and extensions: 7
Failed: 5

Details of failed conformance tests

6.4.3.3-1 — Test does not conform to current ISO standard.

6.6.3.1-1 — Test does not conform to current ISO standard.

6.9.4-4 — Test does not conform to current ISO standard.

6.9.4-7 — Test does not conform to current ISO standard.

Deviance Tests

Number of tests attempted: 94
Passed: 67
Failed due to restrictions and extensions: 23
Failed: 4

Details of failed deviance tests

6.1.5-6 — Test does not conform to current ISO standard.

6.4.6-11 — No check for fields of type file.

6.6.1-6 — No check for procedures or functions that are declared FORWARD but are never defined.

6.6.2-5 — No check to verify that the function identifier is defined within the function.

Error Handling Tests

Number of tests attempted: 46
Passed: 7
Failed due to restrictions and extensions: 39
Failed: 0

Implementation Defined Tests

Number of tests attempted: 15
Passed: 12
Failed due to restrictions and extensions: 1
Failed: 2

Details of failed implementation defined tests

6.11-2 — Alternate operators not allowed.

6.11-3 — Alternate operators not allowed.

Quality Measurement Tests

Number of tests attempted: 23
Passed: 21
Failed due to restrictions and extensions: 2
Failed: 0

Extension Test

Otherwise is implemented as described in the current ISO standard.

PUG

IBM 370 (AAEC)

Pascal 8000 Version 2.0 Validation Suite Report

IBM 370 (AAEC)

Validation Suite Results

Pascal Processor Identification

Computer: IBM 370/168, Model 3
Processor: Pascal 8000, Version 2.0 (27JUL80)

Test Conditions

Tester: Joseph A. Miner, Cornell Computer Services
Date: July 1980
Validation Suite Version: 2.2

Note: In the body of this report, the words "ISO Draft Standard Pascal" and "the ISO Draft Standard" refer to the Draft Pascal Standard ISO DP/7185 published in the April 1980 issue of Sigplan Notices and the May 1980 issue of Pascal News.

Conformance Tests

Number of tests passed: 126 (2 were repaired)
Number of tests failed: 3 (1 basic cause)
Invalid tests discovered: 10

Details of Repaired Tests:(These tests passed after the errors noted were fixed.)

Test 6.6.1-6 was missing a semicolon in the main program after the call of procedure one.

Test 6.6.3.3-3 had type compatibility errors because of anonymous pointer types.

Details of failed tests: Tests 6.4.3.5-2, 6.4.3.5-4, and 6.9.1-1 fail because OS/360 requires that at least one data character be written on each line of a text file (two if the file contains ASA control characters). Zero length records may not be written.

Details of invalid tests: Tests 6.1.2-3 and 6.3-1 require that identifiers that are identical in the first eight characters be distinguished. Both tests passed after the identifiers were changed.

Test 6.1.8-3 shows that either form of comment delimiter may end a comment, as specified by the ISO Draft Standard.

Test 6.4.3.5-1 contains an invalid file type declaration ("file of ptrtoi", where ptrtoi is a variable name, not a type).

Test 6.5.1-1 attempts to define a file of files.

Tests 6.6.3.1-1 is invalid since one of the actual parameters is not of the same type as the corresponding formal variable parameter.

Test 6.6.3.1-5 contains invalid syntax for an actual procedure parameter.

Test 6.6.3.4-2 contains invalid syntax in a formal procedural parameter specification.

Test 6.9.4-4 compares a line previously written to a string constant. The string constant does not match the format used to write the line. (The test succeeds if appropriate changes are made to the program.)

Test 6.9.4-7 expects boolean values to be left justified when written to a text file. The ISO Draft Standard specifies that writing a boolean value to a text file is equivalent to writing the string 'true' or 'false'. Therefore the values should be right justified.

Note: Several tests contain declarations of identifiers that are identical in the first eight characters (6.1.2-3, 6.3-1, 6.4.5-5, and 6.8.2.2-2). Because the Validation Suite assumes that the processor only need distinguish identifiers that differ within the first eight characters, these tests have been reported here as "Invalid Tests". A more recent version of the ISO Draft Pascal Standard (ISO DP/7185) appears to require that a conforming processor distinguish identifiers that differ in any character position.

Deviance Tests

Number of deviations correctly detected: 87
Number of tests not detecting erroneous deviations: 3 (1 basic cause)
Number of tests showing extensions:2
Invalid tests discovered: 3

Details of extensions: **Test 6.8.3.5-12** shows that subrange-like lists are allowed as case-constant elements.

Test 6.8.3.5-14 shows that the "otherwise" clause is allowed in case statements.

Details of deviations not detected: **Tests 6.8.2.4-2, 6.8.2.4-3,** and **6.8.2.4-4** show that it is possible to branch into if statements, between branches of a case statement, and into a case statement.

Details of invalid tests: **Test 6.1.5-6** shows that lower case 'e' is allowed in an unsigned-real number, as specified by the ISO Draft Standard.

Test 6.2.1-5 contains a label that is declared but never defined or referenced. This is allowed in the current version of the Standard. (The compiler issues a warning message in this case.)

Test 6.4.5-5 declares identifiers that are not unique over the first eight characters. The deviation is correctly detected if appropriate changes are made to the identifiers.

Error handling

Number of errors correctly detected: 30
Number of errors not detected: 16 (7 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** show that the variant fields of a record are not "undefined" when the tag field value is changed.

Test 6.4.3.3-12 shows that assignment of an uninitialized empty record is not detected.

Test 6.4.6-7, 6.4.6-8, and **6.7.2.4-1** show that assignment of a set expression containing elements that are not within the subrange base-type of the destination set is not detected if all the elements of the expression set have ordinal values in the range 0..63.

Tests 6.6.5.2-6 and **6.6.5.2-7** show that a file variable may be modified while the associated buffer variable is an actual variable parameter.

Tests 6.6.5.3-5 and **6.6.5.3-6** show that a variable may be DISPOSED while it is an actual variable parameter.

Tests 6.6.5.3-7, 6.6.5.3-8, and **6.6.5.3-9** show that variables created by the variant form of NEW may be used in expressions and on the left hand side of assignment statements.

Implementation Defined

Number of tests run: 15
Number of tests repaired: 1
Details of repaired test: **Test 6.8.2.2-2** contains type compatibility errors caused by anonymous pointer types.

Details of implementation-dependence: **Test 6.4.2.2-7** shows maxint to be 2147483647.

Test 6.4.3.4-2 shows that a set of char is allowed.

Test 6.4.3.4-2 shows that sets must be of 64 elements or less, with sets of integers falling in the range 0..63.

Test 6.6.6.1-1 shows that standard functions may not be used as actual function parameters.

Test 6.6.6.2-11 displays some characteristics of the floating-point arithmetic. The results are reproduced in section 2 of this report. ("Floating-Point Arithmetic Characteristics", below).

Tests 6.7.2.3-2 and *6.7.2.3-3* show that boolean expressions are completely evaluated in all cases.

Tests 6.8.2.2-1 and **6.8.2.2-2** show that the variable on the left-hand side of an assignment statement is selected before evaluation of the expression on the right hand side.

Test 6.9.4-5 shows that two digits are written in an exponent.

Test 6.9.4-11 shows that the default field widths for output are integer — 12; Real — 24; Boolean — 4 if true, 5 if false.

Test 6.10-2 shows that the operation REWRITE(OUTPUT) is permitted

Tests 6.11-1, 6.11-2, and **6.11-3** show that alternative comment delimiters, as well as the symbols (. .) and ¬ are implemented. (Also implemented are the symbols ¬= & and |)

Quality Measurement

Number of tests run: 23 (8 modified)

Results of tests: **Test 5.2.2-1** shows that different identifiers that do not differ in the first eight characters are not flagged.

Test 6.1.3-3 shows that identifiers are distinguished only over eight characters.

Test 6.1.8.4 shows that a semicolon or open comment symbol within a comment is flagged with a warning message.

Tests 6.2.1-8, 6.2.1-9, and **6.5.1-2** show that long lists of types, labels, and variables are allowed in their respective definition parts.

Test 6.4.3.2-4 shows that array [integer] is not allowed.

Test 6.4.3.3-9 shows that variant fields of a record type are overlaid in the order of definition.

Test 6.4.3.4-5 (Warshall's algorithm) uses 0.134 seconds of processor time with all execution tests enabled, and 0.067 seconds without tests. (By comparison, the program uses 0.816 seconds on a B6700 with the Tasmania compiler).

Test 6.6.1.7 shows that five levels of procedure or function nesting is allowed.

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** show that the sqrt, arctan, exp. ln, and sin/cos functions are implemented without any significant error. (Details in section 2 of this report, below.)

Test 6.7.2.2-4 shows that division by and into negative operands is implemented consistently, that the quotient is trunc(a/b) for negative operands, and that mod yields remainder of div with negative operands.

Test 6.8.3.5-2 shows that unreachable case branches are not flagged.

Test 6.8.3.9-8 shows that at least 256 branches are allowed in a case statement.

Test 6.8.3.9-18 is not relevant, since use of a for statement control varible after termination of the loop is detected as an error.

Test 6.8.3.9-20 shows that for statements may be nested at least 15 levels.

Test 6.8.3.10-7 shows that with statements may be nested at least 15 levels.

Test 6.9.4-10 shows that output is flushed at the end of the job.

Test 6.9.4-13 shows that recursive I/O to the same file is allowed.

Details of Modifications

Test 6.4.3.4-5 was modified to use the Pascal 8000 CLOCK function to calculate the processor time used by the program.

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 **6.6.6.2-11** were modified to disable arithmetic interrupts during execution. These tests generate exponent underflow interrupts that are normally trapped and treated as an error.

Test 6.9.4-14 was modified to remove the undeclared and unused file F from the program statement parameters.

Compilation Speed

Several programs were compiled on the IBM 370/168 Model 3 processor using the VM/370-CMS operating system release 6.8 with Basic System Extensions release 2. The virtual CPU times used to compile the programs were recorded. CPU times include time spent interpreting the PASCAL EXEC command file and compiler program loading and initialization. The version of the compiler used was compiled with all execution tests disabled and without any traceback information.

Five programs containing a total of 13,875 lines of code, ranging from 1829 to 3706 lines each, were compiled. When the programs were stored in files containing variable length records with trailing blanks removed, compilation speed was about 30,000 lines per minute, with a range of 23,000 to 45,000 lines per minute. The average speed was around 1,030,000 characters per minute.

When the programs were reformatted in files with fixed-length 80-byte records, compilation times decreased about 2%. The average number of lines per minute increased slightly to 31,400, and the number of characters per minute increased about 145% to 2,-510,000. This increase in speed is apparently due to reduced CMS overhead when processing files with fixed-length records, and high-speed skipping of blank characters by the compiler.

Floating Point Arithmetic Characteristics

Several of the Validation Suite programs test the quality of the floating point arithmetic and mathematical functions. These results are summarized here.

The programs were written by W. J. Cody of Argonne National Laboratory and revised for Pascal by R. A. Freak, University of Tasmania. Parts of the programs are based on an algorithm by M. Malcolm (CACM 15 (1972), pp. 949-951), with some of the modifications suggested by M. Gentleman and S. Marovich, (CACM 17 (1974), pp. 276-277).

Machine Characteristics

Radix of Representation	Beta = 16
Number of base Beta digits in significand	T = 14
Chopping is used (not rounding)	Rnd = 0
More than T base Beta digits participate in post normalization after multiplication	NgRd = 1
Number of bits in exponent representation	IExp = 7
Smallest positive number s.t. 1+ eps<>1	Eps = 2.2204e-16
Smallest positive number s.t. 1-EpsNeg<>1	EpsNeg = 1.3878e-17
Smallest positive number	XMin = 5.3976e-79
Largest positive number	XMax = 7.2370e+75

Arithmetic Function Quality

In the twelve quality tests, various identities were tested with 2000 arguments randomly chosen from a logarithmic distribution over the stated range. The tests are identified by the following numbers:

```
1. sqrt(x*x) - x = 0.0
2. arctan(x) = truncated taylor series.
3. arctan(x) = arctan(1/16) + arctan((x-1/16)/(1+x/16))
4. 2 * arctan(x) = arctan(2x/(1-x*x))
5. exp(x - 0.0625) = exp(x)/exp(0.0625)
6. exp(x - 2.8125) = exp(x)/exp(2.8125)
7. ln(x) = Taylor series expansion of ln(1+y)
8. ln(x) = ln(17x/16) - ln(17/16)
9. ln(x) = ln(11x/10) - ln(11/10)
10. ln(x*x) = 2 * ln(x)
11. sin(x) = 3*sin(x/3) - 4*sin(x/3)**3
12. cos(x) = 4*cos(x/3)**3 - 3*cos(x/3)
```

Table format

From left to right: the test number, the argument range, the number of times the result was too large or too small, the mean relative error in decimal and hexadecimal, the maximum relative error and the argument value at which it occured, and the root-mean-square error in decimal and hexadecimal. (See Table I on following page.)

Modifications to the Validation Suite

The following modifications were made to the test programs before they were processed.

Table I

Test	Range	Large Small	Mean Relative Error	Max Relative Error, @ arg	RMS Relative Error
1	2.5000e-01 1.000e+00	0 919	-1.7565e-17 16**-13.92	1.1077e-16 @2.5058e-01	2.9011e-17 16**-13.73
1	1.0000e+00 4.0000e+00	0 0			
2	-6.250e-02 6.2500e-02	29 0	3.4137e-19 16**-15.34	6.0492e-17 @1.4339e-02	3.2065e-18 16**-14.53
3	6.2500e-02 2.6795e-01	1438 0	7.6733e-17 16**-13.38	2.2153e-16 @1.2595e-01	9.8159e-17 16**-13.29
4	2.6795e-01 4.1421e-01	1776 42	4.6553e-17 16**-13.56	1.5834e-16 @3.1675e-01	5.5639e-17 16**-13.50
4	4.1421e-01 1.0000e+00	359 193	4.0689e-18 16**-14.44	2.2186e-16 @5.4685e-01	5.5398e-17 16**-13.50
5	-2.8407e-01 3.4657e-01	728 181	-4.0753e-17 16**-13.61	2.2279e-16 @5.9158e-02	8.7417e-17 16**-13.34
6	-3.4657e+00 -1.4002e+02	579 207	-2.2268e-17 16**-13.83	2.2048e-16 @-8.8676e+01	5.8089e-17 16**-13.48
6	6.9315e+00 1.7457e+02	576 198	-2.3871e-17 16**-13.80	2.2198e-16 @1.0817e+02	6.0656e-17 16**-13.47
7	9.9998e-01 1.0000e+00	550 414	-2.0664e-17 16**-13.86	4.3116e-16 @1.0000e+00	4.1493e-17 16**-13.60
8	7.0710e-01 9.3750e-01	0 757	-1.0709e-17 16**-14.09	2.1473e-16 @9.3741e-01	6.3062e-17 16**-13.45
9	3.1623e-01 9.0000e-01	0 1072	-1.0710e-17 16**-14.09	3.7234e-16 @8.9347e-01	9.9664e-17 16**-13.29
10	1.6000e+01 2.4000e+01	1503 0	2.2328e-17 16**-13.83	1.1152e-16 @1.9817e+01	3.1590e-17 16**-13.70
11	0.0000e+00 1.5708e+00	1676 122	7.1044e-17 16**-13.41	3.6831e-16 @1.8953e-01	9.4375e-17 16**-13.31
11	1.8850e+01 2.0420e+01	304 1662	-6.2493e-15 16**-11.80	2.7247e-12 @1.9850e+01	6.6908e-14 16**-10.94
12	2.1991e+01 2.3562e+01	1891 93	5.6811e-15 16**-11.83	1.3817e-12 @2.3561e+01	4.0160e-14 16**-11.13

Character Set Changes

- The curly brace ('{' and '}') characters were changed from standard EBCDIC to the text printing (TN) character set.
- The EBCDIC not symbols used for the Pascal up-arrow character were changed to '@'.

Several changes were needed so the file of test programs could be processed by the skeleton program supplied with the test suite.

- Sequence numbers were removed and all lines were truncated to 72 characters or less.
- The heading comment of test 6.8.3.4-1 was missing a comma after the test number, which caused the skeleton program to stop.
- **Test 6.6.1-7** and **6.6.5-3** fail to end with a line containing 'end.' written in lower case in columns one through four.
- The last line in the file is not a complete heading comment with a test number of 999 (it consists only of '{T999'). The skeleton program failed to stop correctly at the end of the file.

Additional repairs made to individual programs are noted in the Validation Suite Report. These repairs deal with programming errors or similar problems. **PUG**

Pascal 1100

Pascal Processor Identification

Computer: Univac 1100/60

Processor: Pascal 1100 — Enhanced descendant of U.S. Naval Ocean Systems Center compiler developed by M.S. Ball

Version 2.1FLR, Updated 10/26/81

Note: This is not a Univac supported product. However, versions of it are available through the University of Maryland.

Test Conditions

Tester: I. L. Ruben ("unofficial" maintainer of the compiler)

Date: October 1981

Validation Suite Version: 2.2

Conformance Tests

Number of tests passed: 125

Number of tests failed: 14 (10 basic causes)

Details of failed tests: **Test 6.2.2.3** contains a scope error which is not detected by the compiler.

Test 6.2.2.8 fails because the compiler restricts assignment to a function identifier to that function's block level.

Tests 6.4.2.2-5 and **6.4.2.2-6** fail because the expression is too long for the code generation scheme utilized. Note however, that the ASCII collating sequence is used, so that these tests would pass if the IF statements were broken up.

Test 6.4.3.5-1 fails because the compiler only allows a file declaration consisting of a file of type. The test has a file of variable (????).

Tests 6.4.3.5-2, 6.4.3.5-3, 6.9.1-1, and **6.9.4-4** fail because characters are written to 1100 text files in multiples of 4, padding with blanks if necessary. Thus, the eoln and eof functions do not occur where expected in these tests.

Test 6.5.1-1 fails because a file of a type, where the type contains (or is) a file type, are not permitted by the compiler (i.e., a file of files is not supported).

Test 6.6.5.2-3 fails because a reset is not allowed on a file that was never written to.

Test 6.8.3.9-7 fails due to a infinite loop introduced by bad code generation in loop termination tests involving maxint.

Test 6.9.4-6 fails because a string is always entirely displayed, even if its field width is smaller.

Test 6.9.4-7 fails because TRUE is right justified.

Deviance Tests

Number of deviations correctly detected: 54

Number of tests showing true extensions: 9

Number of tests not detecting erroneous deviations: 31 (13 basic causes)

Details of extensions: **Tests 6.1.5-6** shows that lower case e may be used in real numbers (e.g. 12.34e-12).

Tests 6.1.7-4, 6.1.7-9 (cases 1 to 4), **6.1.7-10,** and **6.4.5-11** show that a right-hand side string constant (or value procedure parameter) is made the same length (padded with blanks or truncated) as the left-hand side (or formal parameter). In other words, string constants are made to conform across binary operators and assignment.

Test 6.4.2.4-2 shows that real constants are permitted in a subrange declaration.

Tests 6.8.3.5-12 and **6.8.3.5-13** show that a subrange used for a CASE tag is accepted. Also, overlapping and duplicate ranges are detected.

Test 6.10-1 shows that "output" is a predeclared file (note, "input" is also).

Details of deviations not detected: **Test 6.1.2-1** shows that the reserved word NIL may be redefined.

Test 6.1.7-6 shows 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, 6.2.2-7, 6.3-6, and **6.4.1-3** contain a scope error which is not detected by the compiler.

Tests 6.4.5-2, 6.4.5-4, 6.4.5-5 and **6.4.5-13** indicated that type compatibility is used with VAR parameters rather than enforcing identical types.

Test 6.6.2-5 shows that a function without an assignment to the function variable in its block compiles and runs.

Tests 6.6.3.5-2, 6.6.3.6-2, and **6.6.3.6-4** fail because parameter base types are the same (integer).

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 6.8.3.5-10 and **6.8.3.5-11** show that the compiler accepts case tags which are the same type as the index, although a real index is flagged as an error.

Tests 6.8.3.9-2, 6.8.3.9-3, 6.8.3.9-4, 6.8.3.9-16, and **6.8.3.9-19** show that an assignment to a FOR control variable is permitted within the loop.

Tests 6.8.3.9-9 and **6.8.3.9-14** show that the FOR control variable may be declared anywhere, so long as it is declared at the same or outer block (this excludes formals, pointers, and record components).

Test 6.9.4-9 indicates that 0 and negative field widths may be used in a write statement.

Test 6.10-3 shows that "output" can be redefined and yet still be used as the default file for write statements (similarly for "input").

Error Handling

Number of errors correctly detected: 18

Number of errors not detected: 28 (13 basic causes)

Details of errors not detected: Test 6.2.1-7 shows that local variables are not preset to "undefined".

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.

Tests 6.4.3.3-12 shows that an assignment to an empty record is not detected.

Tests 6.4.6-7, 6.4.6-8, and **6.7.2.4-1** indicate that no bounds or overlap 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 and **6.6.5.2-2** fail because I/O has not been implemented according to the standard. Also, characters are written to 1100 text files in multiples of 4, padding with blanks if necessary. Thus, eoln and eof do not occur where expected in the tests.

Tests 6.6.5.2-6 and **6.6.5.2-7** fail because the compiler does not detect (invalid) operations on buffer variables passed as a procedure or function parameter or changed within the range of a WITH statement.

Tests 6.6.5.3-3 and **6.6.5.3-4** fail because dispose (under full memory management support) "ignores" pointers that do not point to the heap (NIL has the value 0). Note, Pascal 1100 supports three levels of memory management configurations (under user option). Under the other two configurations, these tests pass.

Test 6.6.5.3-5 fails because dispose successfully releases the space allocated by new (is this test wrong ????).

Test 6.6.5.3-6 shows that no check is performed for scoping on the parameter to dispose.

Tests 6.6.5.3-7, 6.6.5.3-8, and **6.6.5.3-9** fail because no checks (other than type compatibility) are done on the pointer assignments tested. A check is done, however, that a pointer points to the area allocated to it by new.

Tests 6.6.6.3-2, 6.6.6.3-3, 6.7.2.2-6, and **6.7.2.2-7** fail because overflows are not detected. The values eventually go negative due to the overflow.

Tests 6.8.3.9-5, 6.8.3.9-6, and **6.8.e.9-17** show that a FOR control variable is not invalid after execution of the FOR loop.

Implementation Defined

Number of tests run: 15

Number of tests incorrectly handled: 0

Details of implementation-dependence: **Test 6.4.2.2-7** shows maxint to be 34359738367.

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. Set bounds allowed are 0 to 143.

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 (e.g., single precision reals in range 1.47E-39 to 1.70E+38).

Tests 6.7.2.3-2 and **6.7.2.3-3** show that boolean expressions are evaluated only to the extent needed to determine the result.

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 the exponent field on output is 2, and the real, integer, and boolean default field widths are all 12.

Test 6.10-2 shows a rewrite on the standard file "output" is not allowed.

Tests 6.11-1, 6.11-2, and **6.11-3** show that alternative comment delimiters have been implemented, as have the alternative pointer symbol ("@"). No other symbols from these tests are accepted.

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 is 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.

Test 6.4.3.2-4 shows an array with an integer index type is not permitted.

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 350 ms. to run (on 1100/60).

Test 6.6.1-7 shows that procedures cannot be nested to a level greater than 9.

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, arctan, exp, sin/cos, and ln functions respectively. No significant errors were reported.

Test 6.6.6.2-9 (sin/cos) produced an "out of range" runtime error on the last test in the program. The argument was outside the acceptable range allowed by the 1100 math library for the sine function.

Test 6.7.2.2-4 shows that mod and div are consistent for negative operands except that when i mod 2

(i<0) is 0, it is represented as a negative 0 on the 1100. Thus the expression i-i div 2*2 fails to compare with i mod 2 for the even cases of negative i. Mod returns 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 type 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 shows that the compiler ensures that because the FOR control variable is available after the FOR loop, the final value is the final value of the loop (not 1 greater or less). Thus the range checks (always generated) in the CASE accept the CASE index (value is "pink").

Tests 6.8.3.9-20 and **6.8.3.10-7** indicate the 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 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 the compiler does not accept OTHERWISE in the syntax given in the test. However, it does accept OTHERWISE (and ELSE) when used in the syntax of a CASE label. Further, many other (non-standard) extensions are provided to allow Pascal 1100 to be used for implementation purposes on the 1100. These include external compilations, external variables, 1100 Exec 8 support, and variable length strings. **PUG**

IBM 4341

Computing Services Centre

1st March 1982

Dear Sir,

Enclosed are reports on running the Sale Pascal validation suite against the Pascal compilers on the IBM 4341 (Pascal/VSR 2.0) and the VAX 11/780 (VAX 11 Pascal V 1.2-82). The latter is a later release than the one reported in Pascal News No. 19.

You may wish to publish these in Pascal News.

Yours sincerely,

C.R. Boswell
Director

Pascal processor identification.

Computer: IBM 4341.
Location: Victoria University of Wellington, New Zealand.
Processor: PASCAL/VS R2.0

Test conditions.

Tester: R. H. Hefford (CSC programmer).
Date: January, 1982.
Validation Suite Version: 2.0

Notes:

1) The LANGLVL(STANDARD) option was used with the compiler.

2) The compiler was running under the CMS operating system.

Implementation defined

Number of tests run: 15

Test 6.4.2.2-7 — MAXINT = 2147483647.

Test 6.4.3.4-2 — Implementation allows a set of char.

Test 6.4.3.4-4 — The ord of all set members must be in the range 0. .255.

Test 6.6.6.1-1 — Standard functions are not permitted as parameters.

Test 6.6.6.2-11 — Smallest positive real number larger than zero is 5.39760535E-79. Largest real number is 7.23706558E+75. Reals have a 7 bit exponent and a 14 digit base 16 mantissa.

Test 6.8.2.2-1 — In the situation array[exp2] := exp1; exp1 is evaluated before exp2.

Test 6.8.2.2-2 — In the situation p~ := exp; the expression is evaluated before the position of p~ is evaluated.

Test 6.9.4-5 — Number of digits in exponents is 2.

Test 6.9.4-11 — Default field width for integers, reals and booleans is 12, 20 and 10 respectively.

Test 6.10-2 — A rewrite is allowed on the file output.

Test 6.11-1 — '(*' and '*)' are allowed to delimit comments.

Test 6.11-2 — Alternative symbols are not implemented. '@' is used instead of '^'.

Quality tests

Number of tests run: 24

Number of tests failed: 5

Test 5.2.2-1 Failed: Meaning of the program was changed by the truncation of identifiers.

Test 6.1.3-3 Passed: Number of significant characters in identifiers is 16.

Test 6.1.8-4 Passed: The compiler will help in the discovery of unclosed comments by issuing a warning if it finds inside the comment the start of another comment.

Test 6.4.3.2-4 Failed: The declaration 'everything = array [integer] of integer' is not allowed because there are too many elements.

Test 6.4.3.4-5 Passed: Execution time for the Warshall algorithm was 0.2 seconds. According to the manual the space required would have been 5120 bits or 640 bytes.

Test 6.6.1-7 Failed: Procedures cannot be nested more than 8 levels deep.

Conformance tests

Number of tests run: 138

Number of tests failed: 11

Test 6.1.8-2 Failed: A opening curly bracket in a comment is not allowed.

Test 6.1.8-3 Failed: The closing comment delimiter does not have to be of the same type as the opening one.

Test 6.2.2-1 Passed: The identifier name range appeared to have some special meaning to the compiler and the program did not compile till it was changed to scope.

Test 6.4.3.3-1 Failed: A record declaration of the form d = record; end; was not accepted by the compiler.

Test 6.4.3.5-1 Passed: Error in the program var ptrtoi :~i; instead of type ptrtoi = ~i;

Test 6.4.3.5-2 Failed: Writing an empty line to a file results in a blank followed by an end of line marker.

Test 6.6.3.4-2 Failed: A routine passed as a parameter must not be nested within another routine.

Test 6.6.5.2-3 Failed: Does not seem possible to create an empty file under CMS.

Test 6.7.2.2-5 Failed: The expression (maxint — (maxint div 2)) * 2 was flagged as causing fixed point overflow.

Test 6.8.3.8-2 Failed: A while loop containing no statements is not allowed.

Test 6.9.4-4 Failed: Conforms to the standard except when the number will not fit in the field width specified.

Examples: (_ represents a blank)
Format 0.0:6 Output __0.0 instead of ___0.0
Format 1.0:6 Output _1.E+00 instead of ___1.0
Format 123.456:7:3 Output 123.456 instead of _123.456

Test 6.9.4-7 Failed: Writing of booleans does not conform to the standard. According to the standard the output should have been left justified but the PASCAL/VS output was right justified.

Test 6.9.6-1 Failed: Page procedure did not cause a page throw when writing to a terminal. It will work when writing to a file if the file has the correct format.

Error handling.

Number of tests run: 46

Number of tests failed: 17

Test 6.2.1-7 Failed: The compiler does not check for undefined variables.

Test 6.4.3.3-5 Failed: A change of variant occured in a record (by assigning a value associated with the variant to the tag field). This caused a previous field to cease to exist. A reference to that field did not cause an error.

Test 6.4.3.3-6 Failed: A reference to a field with the undefined value did not cause an error. The undefinition arose because a change of variant occured and those fields associated with the new variant come into existence with undefined values.

Test 6.4.3.3-7 Failed: A reference to an undefined field did not cause an error. In this case the variant changes occured implicitly as a result of assignment to fields.

Test 6.4.3.3-8 Failed: As for 6.4.3.3-7 except no tag field is used.

Test 6.4.3.3-12 Failed: Allowed assignment of an undefined empty record. A contradiction in that the program did not detect the error and printed pass.

Test 6.4.6-5 Failed: An expression with the value 10 was passed to a procedure when the parameter was declared to be 0. .5. The error was not detected.

Test 6.6.5.2-2 Failed: Read past eof not detected.

Test 6.6.5.2-6 Failed: Changing the position of the file variable while it was the actual parameter to a procedure did not cause an error.

Test 6.6.5.2-7 Failed: Changing the file pointer while it is within a with statement does not cause an error.

Test 6.6.5.3-5 Failed: A variable which was an actual variable parameter was refered to by the pointer parameter of dispose without causing an error.

Test 6.6.5.3-6 Failed: A variable which was an element of a record variable list of a with statement was refered to by the pointer parameter of dispose without causing an error.

Test 6.6.5.3-7 Failed: A variable created by the using the variant form of new is used as an operand in an expression. The error is not detected.

Test 6.6.5.3-9 Failed: A variable created by using the variant form of new is used as an actual parameter. The error was not detected.

Test 6.7.2.2-6 Passed: The expression (maxint — (maxint div 2)) * 2 could not be compiled. Other methods were used to get a fixed point overflow and the error was detected.

Test 6.7.2.2-7 Passed: Same problem as for 6.7.2.2-6.

Test 6.8.3.9-5 Failed: The use of a control variable of a for loop after that loop had completed was not flagged as an error.

Test 6.8.3.9-6 Failed: The use of a control variable for a loop which had not been entered was not flagged as an error.

Test 6.9.2-5 Failed: Reading 'ABC123.456' into a real variable did not cause an error message. The result was zero. **PUG**

VAX 11-780

Pascal processor identification

Computer: VAX/11-780

Location: Victoria University of Wellington, New Zealand.
Processor: VAX-11 PASCAL V1.2-82

Test conditions

Tester: R. H. Hefford (CSC programmer).
Date: February, 1982.
Validation suite version: 2.0

Notes

1) The validation suite was compiled using the /CHECK and /STANDARD options.

2) Changes from VAX 11 Pascal V1.0-1 (as reported in Pascal News No., 19.):
   a) Empty record is implemented.
   b) Tag field redefinition allowed.
   c) Run time checking of the appropriateness of the value of variables. Range checks are done for array subscripts, assignment statements, PRED, SUCC, CHR, case selecters and set operations.
   d) Default field width for a boolean is now 7 characters (was 16).

CONFORMANCE TESTS

Number of tests run: 138
Number of tests failed: 11

Test 6.1.3-1 — The compiler issues a warning if an identifier exceeds 19 characters but the program will still run.

Test 6.5.1-1 — Would not allow a file of files.

Tests 6.6.3.1-5, 6.6.3.4-2 — The tests could not be run as this pascal does not allow a procedure passed as a parameter to have a parameter list.

Test 6.6.5.2-3 — A RESET on a non existant file caused the program to fail.

Test 6.6.6.2-3 — The EXP function failed the accuracy test. It gave the value of EXP(9) as 8103.083984. The test program expected a value between 8103.08392 and 8103.08393.

Test 6.8.3.5-4 — Case label ranges exceeding 1000 are not allowed.

Test 6.8.3.9-7 — A for loop with an upper limit of maxint caused overflow to occur.

Test 6.9.4-3 Passed. — The test program had to be modified as the compiler would not accept a packed array of char as a parameter in a readln statement.

Test 6.9.4-4 — When writing real numbers the program used exponential format when the number overflowed the field. The validation suite expected fixed point format.

Test 6.9.4-7 Failed writing booleans. — The program wrote 'TRUEFALSE' and the validation suite expected 'TRUE FALSE'.

Test 6.9.5-1 — Parameter to a read cannot be the element of a packed structure.

DEVIANCE TESTS

Number of tests run: 95
Number of tests failed: 29

Test 6.1.2-1 NIL is not implemented as a reserved word.

Test 6.1.5-6 'e' is equivalent to 'E' in real numbers.

Test 6.2.2-4 — Allowed a global symbol to be used within a procedure with its global definition and then allowed it to be redefined.

Test 6.3-6 — A constant was used in its own declaration.

Test 6.4.1-2 — The compiler allowed the use of types in their own declaration.

Test 6.4.1-3 — Again a type was used in its own definition. In this case a global symbol was available with the same identifier.

Test 6.4.5-2 thru **6.4.3-5, 6.4.5-13** — The compiler checks the types of the formal and actual parameters. The identifiers do not have to be the same.

Test 6.6.2-5 — Functions do not have to contain an assignment to the function name identifier.

Tests 6.6.3.5-2, 6.6.3.6-2 thru *6.6.3.6-5* — These tests could not be run as this pascal does not allow a procedure passed as a parameter to have a parameter list.

Test 6.8.2.4-2 — Jumps between branches of an if statement are allowed.

Test 6.8.2.4-3 — Jumps between branches of a case statement are allowed.

Test 6.8.2.4-4 — Allowed a goto into a case statement.

Tests 6.8.3.9-2, 6.8.3.9-3, 6.8.3.9-4 — Allows assignment to the control variable in a for loop.

Test 6.8.3.9-9 — A non local variable at an intermediate level can be used as a for statement control variable.

Test 6.8.3.9-13 — A formal parameter can be used as a for statement control variable.

Test6.8.3.9-14 — A global variable (at program level) can be used as a for statement control variable.

Test 6.8.3.9-16 — A for statement control variable value can be read during the execution of the for statement.

Test 6.8.3.9-19 — Allowed a nested for loop using the same control variable. In this test the inner for loop is in a procedure called from within the outer for loop.

Test 6.9.4-9 — Allowed the use of a field width of zero and minus one when writing integers.

Error Handling

Number of tests run: 46
Number of tests failed: 18

Tests 6.4.3.3-5 thru **6.4.3.3-8** — Reference to undefined or nonexistant variables was not detected as an error. The variables become undefined or nonexistant due to a change of variant.

Test 6.6.2-6 — Use of a function with an undefined value was not detected.

Test 6.6.5.2-1 — The test could not be carried out because the program would not do a PUT to a file it had just done a RESET on.

Test 6.6.5.2-6 — Changing the current file position of a file f, while the buffer variable is an actual parameter to a procedure was not detected as an error.

Test 6.6.5.2-7 — 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 was not detected.

Tests 6.6.5.3-3 thru **6.6.5.3-6** — DISPOSE accepted as parameter a NIL pointer, an undefined pointer, a pointer that is pointing to a actual variable parameter and a pointer that is pointing to a variable that is an element of a record variable list. No error message or warning was given.

Test 6.6.5.3-7, 6.6.5.3-8, 6.6.5.3-9 — A variable created by the use of the variant form of new is used as an operand in an expression, as a variable in an assignment statement and as an actual parameter. This was not detected as an error.

Test 6.8.3.9-5 — Allowed use of a control variable after the for loop had completed. The variable had retained the final value it had in the for loop.

Test 6.8.3.9-6 — If a for loop is not entered the control variable retains the value it had before the for loop is entered.

Test 6.8.3.9-17 — Two nested for statements can use the same control variable.

IMPLEMENTATION DEFINED

Number of tests run: 15

Test 6.4.2.2-7 — The implementation defined value of maxint is 2147483647.

Test 6.4.3.4-2 — Implementation allows set of char.

Test 6.4.3.4-4 — Set element values must not exceed 255.

Test 6.6.6.2-11

1) The radix of the floating-point representation is
2) The number of base 2 digits in the floating-point significand is 24.
3) The arithmetic rounds.
4) The number of bits reserved for the representation of the exponent of a floating-point number is 8.
5) The exponent of the smallest positive fl. pt. no. is – 128.
6) The exponent of the largest finite floating-point number is 127.
7) The smallest positive floating-point number eps such that 1.0+eps <> 1.0 is 5.96046448E-08.
8) The smallest positive floating-point number is 2.93873588E-39.
9) The largest finite floating-point number is 1.70141173E+38.

Test 6.7.2.3-2 — In the short circuit evaluation of (a and b) both expressions are evaluated.

Test 6.7.2.3-3 — In the short circuit evaluation of (a or b) both expressions evaluated.

Test 6.8.2.2-1 — The binding order of (a[i] := exp) is selection then evaluation.

Test 6.8.2.2-2 — The binding order of (p^:= exp) is selection then evaluation.

Test 6.9.4-5 — The number of digits written in an exponent is 2.

Test 6.9.4-11 — Implementation defined default field width values:

INTEGERS:	10 characters
BOOLEAN:	7 characters
REAL:	16 characters

Test 6.10-2 — A rewrite can be performed on the file output.

Test 6.11-1 — Alternate comment delimiters have been implemented.

Test 6.11-2, 6.11-3 — Alternative symbols not implemented.

Test 6.6.6.1-1 — Test could not be done as this pascal will not accept a function or procedure with a parameter list as a parameter to a function or procedure.

QUALITY

Number of tests run: 23
Number of tests failed: 2

Test 6.1.8-4 — The program contained an unclosed comment bracket. The compiler did not assist in any way with finding this error. The program compiled without errors.

Test 6.4.3.2-4 — Declaration 'array[integer] of integer' is not allowed. The error message was 'Index type must not be integer'.

Test 6.4.3.3-9 — The fields of a record are stored in memory in the order that they are declared.

Test 6.4.3.4-5 — Warshall's algorithm in Pascal. Execution time was 102 milliseconds. **PUG**

Back Issues

1983

Pascal News
2903 Huntington Road
Cleveland, Ohio 44120

Back issues are requested and sent in sets

OUT OF PRINT

$15 ☐ set 1 Issues 9 . . . 12 (September 1977 — June 1978)

$15 ☐ set 2 Issues 13 . . . 16 (December 1978 — October 1979)

$15 ☐ set 3 Issues 17 . . . 20 (March 1980 — December 1980)

$15 ☐ set 4 Issues 21 . . . 23 (April 1981 [mailed January 1982] — September 1981 [mailed March 1982])

Requests from outside USA please add $5 per set.

All memberships entered in 1983 will receive issue 24 and all other issues published in that year.
Make check payable to: "Pascal Users Group," drawn on USA bank in US dollars.

Enclosed please find US $____.__
on check number_____

(I have difficulty reading addresses. Please forgive me and type or print clearly)

My address is:

NAME _____

ADDRESS _____

_____

_____

PHONE _____

COMPUTER _____

DATE _____

JOINING PASCAL USER 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").
- When you join PUG any time within a year: January 1 to December 31, you will receive *all* issues *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.

RENEWING?

- Please renew early (before November) 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*.

ORDERING BACK ISSUES OR EXTRA ISSUES?

- Back issues will have a price rise to $25 on July 83
- Our unusual policy of automatically sending all issues of *Pascal News* to anyone who joins within a year 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*.
- Issues 9 . . 12, 13 . . 16, & 17 . . 20, 21 . . 23 are available from PUG(USA) all for $15.00 a set.
- Extra single copies of new issues (current academic year) are: $10 each — PUG(USA).

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. Please send material single-spaced and in camera-ready (use a dark ribbon and lines 15.5 cm. wide) form.
- All letters will be printed unless they contain a request to the contrary.

IMPLEMENTATION NOTES ONE PURPOSE COUPON

0. **DATE**

1. **IMPLEMENTOR/MAINTAINER/DISTRIBUTOR** *(* Give a person, address and phone number. *)*

2. **MACHINE/SYSTEM CONFIGURATION** *(* Any known limits on the configuration or support software required, e.g. operating system. *)*

3. **DISTRIBUTION** *(* Who to ask, how it comes, in what options, and at what price. *)*

4. **DOCUMENTATION** *(* What is available and where. *)*.

5. **MAINTENANCE** *(* Is it unmaintained, fully maintained, etc? *)*

6. **STANDARD** *(* How does it measure up to standard Pascal? Is it a subset? Extended? How.*)*

7. **MEASUREMENTS** *(* Of its speed or space. *)*

8. **RELIABILITY** *(* Any information about field use or sites installed. *)*

9. **DEVELOPMENT METHOD** *(* How was it developed and what was it written in? *)*

10. **LIBRARY SUPPORT** *(* Any other support for compiler in the form of linkages to other languages, source libraries, etc. *)*

Membership 1983

Pascal News
2903 Huntington Road
Cleveland, Ohio 44120

Please enter my

☐ New or ☐ Renew

membership in Pascal Users Group. I understand I will receive "Pascal News" whenever it is published in this calendar year.

Pascal News should be mailed

1 yr. ☐ in USA $20 ☐ outside USA $30 ☐ AirMail anywhere $55

3 yr. ☐ in USA $40 ☐ outside USA $70 ☐ AirMail anywhere $115

(Make checks payable to: "Pascal Users Group," drawn on USA bank in US dollars)

Enclosed please find US $ ____.____
on check number _____

(Invoice will be sent on receipt of purchase orders. Payment must be received before newsletter will be sent. Purchase orders will be billed $10 for additional work.)

(I have difficulty reading addresses. Please forgive me and type or print clearly.)

My address is:

NAME _____

ADDRESS _____

_____

_____

PHONE _____

COMPUTER _____

DATE _____

☐ This is an address correction here is my old address label:

_____

_____

_____

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 goals and is being used successfully for:

- teaching programming concepts
- developing reliable "production" software
- implementing software efficiently on today's machines
- writing portable software

Pascal implementations exist for more than 105 different computer systems, and this number increases every month. The "Implementation Notes" section of *Pascal News* describes how to obtain them.

The standard reference ISO 7185 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" 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.

Remember, Pascal User's Group is each individual member's group. We currently have more than 3500 active members in more than 41 countries.

FINIS

(FOLD HERE)

PLACE
POSTAGE
HERE

Bob Dietrich
M.S. 92-134
Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077
U.S.A.

(FOLD HERE)

NOTE: Pascal News publishes all the checklists it gets. Implementors should send us their checklists for their products so the thousands of committed Pascalers can judge them for their merit. Otherwise we must rely on rumors.

Please feel free to use additional sheets of paper.

'LEMENTATION NOTES ONE PURPOSE COUPON

POLICY: PASCAL USERS GROUP (15-Sep-80)

Purpose: The Pascal User's Group (PUG) promotes 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 takes 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 "[illegible]" makes it a viable alternative for software production and justifies its further use. We all strive to make using Pascal a [illegible] 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 goals and is being used successfully for:
* teaching programming concepts
* developing reliable "production" software
* implementing software efficiently on today's machines
* writing portable software

Pascal implementations exist for more than 105 different computer systems, and this 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" 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.

Remember, Pascal User's Group is each [illegible] member's group. We currently have more than 3500 active members in more than 40 countries; this year Pascal News is averaging more than 100 pages per issue.

Policy