

Return to:

Pascal News

2903 Huntington Rd. • Cleveland, Ohio 44120

Return postage guaranteed Address Correction requested

BULK RATE
U.S. POSTAGE
PAID
WILLOUGHBY, OHIO
Permit No. 98

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

Pm 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

- *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 Gitron*

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

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 in-

cludes 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
  nil = '0';
begin
  if nil <> '0' then
    writeln( 'WRONG', nil, 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**

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 distribution 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

