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 standardised 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 fail.

And now for something different --
1. To those of you who requested supply of back numbers, I regret that I still haven’t found a solution to the very high direct and indirect costs of small numbers of reprints. Even when you are willing to cover all costs, there remains the burden of manual labour and time. Any ideas?
2. PLEASE, if you must use purchase orders, include include subscription with it. Otherwise there will be a drowning in paperwork.
3. I still haven’t resolved the problem of how to service subscribers to PUG(USA). Unfortunately, if they are serviced out of local funds, the EFFECT (whatever the intention) is subsidisation of more prosperous PUG(USA) by barely solvent PUG(UK) and a detrimental effect on local service.

In a similar vein, escalating costs necessitate an increase in the numbers of subscribers, or, an increase in subscription cost for ‘83. It is suggested that a subscription of 25 Pounds per annum for firms and institutions, with a personal subscription of half that amount, would be equitable,
From Wireless World, reproduced with Phillip Durrington's permission

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

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 by the programme. Mr Callaghan happily talked on the programme about the usefulness of microcomputers to children, and the educational benefits that can be gained by using them. The programme also highlighted the potential of microcomputers in the classroom, and the importance of incorporating them into the curriculum.

In order to promote the use of microcomputers, the programme suggested that schools could use them to enhance the learning experience for students. It was also suggested that teachers should be trained to use microcomputers effectively in the classroom, and that schools should provide adequate resources for this purpose.

The programme also discussed the potential of microcomputers in other areas, such as science and mathematics. It was suggested that microcomputers could be used to help students understand complex concepts more easily, and to make learning more engaging.

The programme concluded by encouraging schools to consider the use of microcomputers in their curriculum, and to provide appropriate training and resources for teachers.

Received a call from Andy Wicket yesterday evening. Apparently there has been another change of US editor. This accounts for the delay in producing PR24. The good news is:

(a) That production of PR24 is now progressing again.
(b) That I will receive a number of copies of back issues of PR.

If you did not receive any editions prior to 22/13, please let me have details plus a large stamped addressed envelope for each copy you need.

With any luck we may end up with a few surplus copies, so if you would like to purchase any, please contact me.

I hope we can now satisfy everyone who was dissatisfied by the very patchy coverage of PR.

Remember the new address and telephone number for M.I.S.S(UK) is:

P.O. Box 52,
Pinner,
Middlesex. HA5 3EE.

Tel: 01 866 3816

There aren't any agreed procedures and deadlines for including European originating material in future editions, but if you have material you would like to publish please send it as soon as possible to the UK address. If for any reason I can't have it included in the next edition, the option is still there to produce further UK supplements.
ProPascal

is a native-code Pascal compiler for Z80-based microcomputers. It is designed to run under CP/M or CDOS, in a memory area of at least 48K RAM. The minimum disc storage required is two 120K drives.

Pro Pascal was produced by Prospero Software of London, England, and released in October 1981. The range of machines on which it is currently running includes:


Pro Pascal is a complete implementation of the recently published BSI/ISO Standard for Pascal, with just two restrictions: conformant array parameters are not included, and files may not be defined within structured- or pointer-types. A number of important extensions have been added to the language, making Pro Pascal suitable for a wide range of applications in the professional, business, scientific or educational sectors.

### Standard features

**Data types:**

- integer: range -2147483647 to +2147483647 (4 bytes)
- real: 7-digit precision, range E-38 to E+38 (4 bytes)
- char: ASCII 128-character set (1 byte)
- boolean: false, true (1 byte)
- enumerated: up to 256 constants (1 byte)
- subranges: including 1- and 2-byte integers
- arrays: any number of dimensions
- records: including variant records
- sets: up to 128 elements (16 bytes)
- pointers: (2 bytes)
- files: text and non-text

**Statements:**

- assignment
- procedure call
- GOTO (including jumps out of blocks)
- compound (BEGIN ... END)
- IF ... THEN ... ELSE ...
- CASE
- REPEAT ... UNTIL ...
- WHILE ... DO ...
- FOR ... TO/DOWNTO ... DO ...
- WITH ... DO ...

**Operators:**

- arithmetic: +, -, *, /, DIV, MOD
- logical: AND, OR, NOT
- comparison: \( \leq, \geq, <, >, \leq, >\)
- set: +, -, * IN \{ ... \}

**Procedures and functions:**

- procedure/function declarations (fully recursive)
- value, VAR, procedural and functional parameters

**Standard procedures:**

- reset, rewrite, get, put
- read, readln, write, writeln, page
- new, dispose, pack, unpack

**Standard functions:**

- abs, sqr, trunc, round, ord, chr
- pred, succ, odd, eof, eoln
- sin, cos, exp, ln, sqrt, arctan

### Extensions

**Data types:**

- string[n]: dynamic-length strings (1 to 255 bytes)
- longreal: 16-digit precision, range E-308 to E+308 (8 bytes)

**Statements:**

- CASE ... OTHERWISE

**Additional procedures:**

- delete, insert, str (string handling)
- assign, close, erase (CP/M file interface)
- update, seek (random-access file handling)
- chain, putcomm, getcomm (program chaining)
- move (assignment without type checking)

**Additional functions:**

- concat, copy, length, pos (string handling)
- fstat (does this CP/M file exist?)
- cstat (has a key been pressed?)
- memavail (how much dynamic storage left?)
- rand (random number generator)

**Lexical enhancements:**

- Source file inserts (\$I filename)
- Identifiers containing underscore (_)
- Hex constants (e.g. 01FFH)
- Longreal constants (e.g. 1.0DO)

---

**Trademarks**

Advantage, North Star: North Star Computers
Apple: Apple Computer
CDOS: Cromemco
CP/M: Digital Research
Pet: Commodore Business Machines
Softcard: Microsoft Consumer Products
Superbrain: Inseric Data Systems
TeleVideo: TeleVideo Systems
Z80: Zilog Corporation
Pascal Users Group (UK),
PO Box 52,
Pinner,
Middlesex. HA5 3FE
UK.
Tel: 01 866 3816

Annual calendar year subscription:
  1982 : £9-00.
  1983 : Individuals - £12-50.
         Institutions - £25-00.

Name :
Address :
Phone :
Computer system :
Special interests:

Please enter me as subscriber for / 1 years.
I enclose £
I attach :

for publication in Pascal News (or
Supplements to PN).
Pascal - an effective language Standard

B A Wichmann, 6/5/82

Article formed the basis of piece in Computer Weekly by Philip Hunter, 11th Feb 1982, page 14

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 microprocessors and mainframes as well. In fact, Pascal is one of the few languages that form a bridge between microprocessor systems and the mainframe 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 standardisation 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 be 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 2 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, 0.123)
else
  writeln('RIGHT')
end.
---

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

---

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 summarised thus:

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Errors detected</th>
<th>Accuracy of error messages</th>
<th>Recovery from last error</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>2.5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>2.5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>3.5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>4.5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>H</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>I</td>
<td>3.5</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

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.
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 which 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 conformance array tests are a collection of conformance, deviance, implementation-defined, implementation-dependent, error-handling and quality tests designed to test the conformance 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:

(i) 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 (e.g., 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 standardised internal representation of the program and will not be affected by legal lexical substitutions. Certain parts of the program may need customisation 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 need 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 violates 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. Further more 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 organisation 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. These 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 standardised format reduces the risk of hidden bias and provides a concise statement of how an implementation has performed under test.

Second the user community can more closely monitor the extent of conformance with the standard.

The Guide specifies ten section headings for the standardised 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 organisation, and will be found helpful by most people undertaking tests of implementations. Once transcription and organisation 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 standardisation of Pascal.
Dear Nick,

Please find enclosed an up to date version of my Pascal Bibliography. Naturally I would not expect any of the readers of Pascal News to have enough spare time to type up these programs so I will gladly let them have tapes in return for postage and administrative costs.

Finally as a self taught programmer I would appreciate comments from readers particularly if anyone finds an example of where I have failed to make use of the facilities available within the language.

THE UNIVERSITY OF NOTTINGHAM

A Pascal Bibliography. by Tony Heyes
Blind Mobility Research Unit,
Department of Psychology,
University of Nottingham.
England.

Introduction.

The Pascal Bibliography is a package of programs written in standard Pascal and should therefore be easily transported. It enables users to store references and to retrieve them either by AUTHOR name or by KEYWORD; or logical combinations of AUTHORS and KEYWORDS. The bibliography is designed for human use; it uses very explicit prompts.

Design Philosophy.

The bibliography consists of a collection of ITEMS. Each ITEM takes the form of:

One line devoted to AUTHOR or ADDRESSEE names.
Two lines devoted to TITLE or ADDRESS.
Two lines devoted to LOCATION.
DATE, ITEM NUMBER.
Two lines devoted to KEYWORDS.

For example:

HEYES A.D., FERRIS A.J., ORLOWSKI R.J.
COMPARISON BETWEEN TWO METHODS OF RESPONSE FOR
AUDITORY LOCALISATION IN THE AZIMUTH PLANE.
J. ACOST. SOC. AMER., 58; 1336-1339 1975

DEAFNESS, LOCALISATION, AUDITORY DISPLAYS
STEREOPHONIC SOUNDS, KINAESTHESIS

If ITEMS are addresses the convention is to store the address on the two lines of title.

Tony Heyes
NEW USERS SHOULD ASK IF THEY MAY HAVE ACCESS TO AN ESTABLISHED BIBLIOGRAPHY AND THEN TRY USING "bibout" TO LOCATE ITEMS OF INTEREST.

To logout respond to the system prompt "$" by typing "control Z".

The Programs.

a) "bibin"

The opening prompt allows the selection of one of the following options:-

APPEND

The prompts should be sufficiently explicit, but note:-

(1) Authors and keywords should be separated by commas. Since they are used in the dictionary they should not spill over the end of a line. They can be any length but only the first 20 characters are significant.

(2) The terminal will probably be set to produce lower case letters. The program will automatically convert them to upper case. If you wish to override this, begin each line of text with a backslash [\].

(3) The date must be a single integer e.g. 1980.

(4) If addresses are to be stored use the two title lines, close pack but indicate new lines with a backslash [\].

(5) A personal local storage reference may be kept on the second location line. It should be enclosed in square brackets; e.g. [BM760] means that a copy of this item is in the BM library, entry number 760.

CHANGE

Answer the prompts but please note of the following:-

1) You must know in advance the ITEM number of the items you require to change.

2) You have to retrieve the ITEMS from the bibliography so CHANGE is relatively slow; be patient. It saves time, if you are changing more than one ITEM to make the changes in numerical order of ITEM number.

3) You retrieve the ITEM to be changed from the bibliography, the changed ITEM goes into the PENDING TRAY. If you change the same ITEM more than once in a single day only the last version will survive.

SPECIAL FACILITY

This option moves the contents of the SCRATCH file into the PENDING tray. It can be used for moving ITEMS from one bibliography to another. Since SCRATCH is a text file, ITEMS may be changed using an editor and then loaded back into the PENDING tray. (Clever stuff!).
b) "bibout"
The computer will count the ITEMS in the bibliography and then offer the option of producing a HARD COPY of the dictionary or doing a SEARCH for ITEMS.

SEARCH
You may either search by NUMBER or, more usually by using the DICTIONARY.
You may opt to send the results either to the TERMINAL or to the SCRATCH FILE for subsequent printing.
SEARCH by NUMBER
The search is terminated by asking to search for item number zero [0].
A block of ITEMS may be searched for by asking to search for item number minus one [-1]. You will then be asked for the lowest and the highest item numbers of the block.
SEARCH by DICTIONARY
You will be asked for a word i.e. an AUTHOR name or a KEYWORD.
The computer will look this up in the DICTIONARY and list the ITEM numbers of all ITEMS containing this word in their AUTHOR or KEYWORD string. If you are doing a single word search answer the next prompt with a full stop [.], and then the instruction to 'LOOK UP'. If, however, it is a multiple word search give the next word. Once again the corresponding ITEM number list will be printed out.
The answer to the prompt "AND, OR, or NOT" enables you to combine the current ITEM number list with the previous ITEM number list. For instance:
- AND Only numbers present in both lists are retained.
- OR All numbers from both lists are retained.
- NOT Numbers present in the previous list are deleted from the current list.
A new current list is printed out showing the results of the selection. The search sequence may be continued for any number of logical combinations of words. At any time a search for the ITEMS in the current list may be initiated by giving a full stop [.]. After which you may either LOOK UP the selected ITEMS or, if you have made a mistake in your list combinations simply RESTART. There is one special word, namely ***, this word will match all the dictionary.

c) "outdict"
No prompts and no option, simply type "outdict" in answer to the system prompt "@" to obtain a hard copy of the current DICTIONARY.
Note, you must have first prepared a copy of the DICTIONARY by running the appropriate HARD COPY option of "bibout".

d) "opr scratch"
This program is run to obtain the printed output from "bibout", provided the option had been chosen to send the output to the SCRATCH FILE.
No prompts and no options, simply type "opr scratch" in answer to the system prompt "@" to obtain a hard copy of the contents of the SCRATCH FILE.
N.B. If you would like to list the SCRATCH FILE to the terminal to check the contents then run "cat scratch".

Acknowledgements.
I gratefully acknowledge the encouragement and support I have received from Roger Henry and Chris Blunsdon. The bibliography was originally intended for use by the members of the BLIND MOBILITY RESEARCH UNIT it is however available to any members of the Pascal Users Group. Would anyone wishing to take up this offer please contact Tony Heyes to arrange medium of transportation.
NOTES FOR IMPLEMENTORS

The following notes outline the steps the implementer should take in order to establish a new bibliography. After this groundwork, the user can use the shell commands bibin, bibout, and outdict to build and manipulate the bibliography.

1. The bibliography system requires 6 workfiles named b1 to b6. The recommended practice is for the user to devote a directory to the bibliography, say 'user/bib'. The workfiles can be created easily using the cat command. E.g.

```
cat > b1
```

File b3 requires a link named scratch. This can be created by the command -

```
l n b3 scratch
```

2. b6 is used as a temporary scratch file during the overnight run. It grows to be as large as b1. If there is insufficient room on the user's disc b6 may be coerced on to another disc.

3. The bib directory must contain the following shell commands:

```
bibin Bibin.out b1 b2 b3 b4 b5
bibout Bibout.out b1 b2 b3 b4 b5
bibupdate Bibupdate.out b1 b2 b3 b4 b5 b6
outdict (1pr b4;rm b4;>b4)6
```

4. Finally, an entry must be made in the UNIX table 'crontab' so that bibupdate will be executed during the night.

```
program Bibin(input,output,bank,dict,scratch,dlist,PendingTray);
(* To ADD, CHANGE or REMOVE items, instructions left in a PendingTray file 'pending', actual changes made by running "Bibupdate.p" *)
(* written by Tony Heyes, Blind Mobility Research Unit, Department of Psychology, The University, Nottingham, U.K. *)
```

```
label 10;
```

```
const LineLn = 70;
RowLn = 20;
Hitag = 10000;
NonDate = -1066;
```

```
type string = packed array [1..LineLn] of char;
item = record
  authors,title1,title2,
  place1,place2 : string;
  date : integer;
  key1,key2 : string
end;
```

```
word = packed array [1..20] of char;
row = array [1..RowLn] of integer;
dic = record
  name : word;
  numbers : row;
  cont : boolean
end;
```

```
TagItem = record
  tag : integer;
  entry : item
end;
```

```
var empty,entry : item;
bank : file of item;
PendingTray,TempPendingTray : file of TagItem;
dlist,scratch : text;
dict : file of dic;
TagEntry : TagItem;
ch,AppendOption,ChangeOption,MainOption,HelpOption,SpecialOption : char;
chge : boolean;
&n,nn,count : integer;
```

```
procedure InlChar (var ch : char);
(* to read the first character of a word typed into the terminal *)
begin
  ch := input;
  while not (ch in ['A'..'Z','a'..'z']) do
    begin (* skips along until first character found *)
      get(input);
      if coin(input)
        then
```

```
```
procedure InlInt(var int1 integer);
(* to read an integer and not cause a fatal error if a character is given *)
var ch : char, a, OrdZero1 integer, NegFound1 boolean.
begin
repeat (* skips along until integer is found *)
  get(input);
  if eoln(input)
    then
      writeln;
      write('ERROR: digit required .... ')
    end;
  ch := input
until ch in ['-','+','0'..'9']
if ch='-' then
  begin
    NegFound1 := true;
    get(input);
    writeln;
    write('ERROR: digit required .... ')
  end;
  ch := input
until ch in ['-','+','0'..'9']
if ch='+' then
  end;
else
  begin
    NegFound1 := false;
    if ch='+' then
      begin
        get(input);
        ch := input
      end;
    a := 0;
    OrdZero1 := ord('0');
    repeat
      a := 10*a+ord(ch)-OrdZero1;
      ch := input
    until not (ch in ['0'..'9'])
    if NegFound1 then
      int := -a
    else
      int := a
  end;
end; (* of InlInt *)

procedure VDUinString(var str : string);
(* to input from terminal *)
var i,n : integer;
   ch : char;
   AllCaps : boolean;
begin
  n := 0;
  AllCaps := false;
  repeat
    n := n+1;
    read(ch);
    if (n=1) and (ch=' ') then
      begin
        n := 0;
        if (n=1) and (ch='\') then
          begin
            (* defeat automatic shift with '\' *)
            AllCaps := false;
            n := 0;
          end;
        if n>0 then
          begin
            if AllCaps then
              begin
                if ch in ['a'..'z'] then
                  ch := chr(ord(ch)-32);
              end;
            str[n] := ch
          end;
        until eoln(input);
        for i=n+1 to LineLn do
          str[i] := ' '
  end; (* of VDUinString *)

procedure ScratchInStr(var str(* input from file scratch *));
var n,i : integer;
   ch : char;
begin
  if not eof(scratch) then
    begin
      n := 0;
      repeat
        read(scratch,ch);
        until (ch=':') or (eof(scratch))
      begin
      end;
    end;

  ch := char;
  repeat
    n := 0;
    read(scratch,ch);
    until (ch=':') or (eof(scratch))
  begin
    n := 0;
    repeat
      read(scratch,ch);
      until (ch=':') or (eof(scratch))
    end;
  begin
    int := -a
  end;
function ScratchHoldsItems : boolean;
  (* to inspect the SCRATCH FILE and check that ITEMS are complete *)
  var count, LineNo : integer;
  FaultFound, HeadingError, NegFound : boolean;
  procedure CheckLine;
    var CharCount : integer;
    LineTooLong, BadLine : boolean;
    begin
      LineNo := LineNo + 1;
      CharCount := 1;
      BadLine := false;
      LineTooLong := false;
      get(scratch);
      while (not eoln(scratch)) and (CharCount < LineLn + 9) do
        begin
          get(scratch);
          CharCount := CharCount + 1;
          if CharCount = 9 and (scratch <> '.') then BadLine := true;
        end;
      if BadLine then
        begin
          if FaultFound then
            writeln('Line', LineNo : 4, ', bad line :'' missing.');
          writeln('Line', LineNo : 4, ', bad line :'' missing.');
        end;
      if LineTooLong then
        begin
          FaultFound := true;
          writeln('Line', LineNo : 4, ', overflow.');
        end;
    end;
  procedure empt; (* to empty an ITEM *)
    var NoChar : string;
    a : integer;
ch : char;
begin
  NoChar[1] := ',
  NoChar[2] := ',
  NoChar[3] := ',
for a:=4 to Lineln do
  NoChar[a] := '.';
with empty do
  begin
    authors := NoChar;
    title := NoChar;
    title2 := NoChar;
    place := NoChar;
    place2 := NoChar;
    date := NonDate;
    key1 := NoChar;
    key2 := NoChar
  end;
for a:=2 to 9 do
  begin
    case a of
      2: ch := ',';
      3: ch := 'e';
      4: ch := 'm';
      5: ch := 'p';
      6: ch := 't';
      7: ch := 'y';
      8: ch := ' ';
      9: ch := ',';
    end; (* of case *)
  end; (* of empt *)
end; (* of GetReference *)
procedure OutRecord(entry: item; n: integer);
(* to write to the terminal *)
var a: integer;
begin
  for a:=1 to 7 do
    begin
      writeln('---------I');
      writeln;
      with entry do
        begin
          writeln(authors);
          writeln(title);
          writeln(title2);
          writeln(place);
          writeln(place2);
          writeln(date);
          writeln(key1);
          writeln(key2);
        end
    end;
  writeln(Item number : ',n :5);
end; (* of OutRecord *)

procedure GetReference(n: integer);
(* to count through bank to find an ITEM *)
begin
  if count <
    then
      begin
        reset(bank);
        count := 1
      end;
  while (count < n) and (not eof(bank)) do
    begin
      count := count +1;
      get(bank)
    end;
  if eof(bank)
    then
      begin
        writeln;
        writeln('You have only got',count -1,' Items.');
        writeln;
        writeln;
      end
    else
      OutRecord(bank',n)
  end; (* of GetReference *)

procedure change(var entry: item; m: integer);
(* to change the mth. ITEM *)
var line: integer;
DMOption, LineOption: char;
str : string;
begin
  writeln;
  writeln;
  repeat
    writeln('Do you wish to DELETE or MODIFY .... ');
    writeln(DMOption),
    until DMOption in ['D','d','M','m'];
  if DMOption in ['D','d']
    then
      begin
       empt,
        entry := empty
      end
    else
      begin
        writeln;
        writeln('You may REPLACE a line,');
        writeln('move to the NEXT line,');
        writeln('or SKIP to the end of the item. ');
        writeln;
        line := 0;
        repeat
          line := line +1;
        end;
with entry do
  case line of
    1: str := authors;
    2: str := title1;
    3: str := title2;
    4: str := place1;
    5: str := place2;
    6: 
    7: str := key1;
    8: str := key2
  end; (* of case *)
if line<>6
then
  begin
    writeln;
    writeln(str);
    writeln(output);
    repeat
      write('REPLACE, NEXT line or SKIP to end .... ');
      InlChar(LineOption)
    until LineOption in ['R', 'r', 'N', 'n', 'S', 's'];
    writeln;
    if LineOption in ['R', 'r']
    then
      begin
        writeln('Type replacement line : ');
        writeln;
        VDUInString(str)
      with entry do
        case line of
          1: authors := str;
          2: title1 := str;
          3: title2 := str;
          4: place1 := str;
          5: place2 := str;
          7: key1 := str;
          8: key2 := str
        end; (* of case *)
    end
  else
  begin
    writeln('Date : ,entry.date :4);
    writeln;
    write('REPLACE, NEXT line or SKIP to end .... ');
    InlChar(LineOption)
    until LineOption in ['R', 'r', 'N', 'n', 'S', 's'];
    if LineOption in ['R', 'r']
    then
      begin
        writeln('Type replacement date : ');
        write(' : ');
        InlInt(entry.date)
        end;
  end; (*of change*)
begin (*MAIN PROGRAM*)
count := HiTag;
n := 1;
reset(PendingTray);
rewrite(TempPendingTray);
begin (*copy down existing contents of file 'PendingTray'*)
  TempPendingTray := PendingTray;
  put(TempPendingTray);
  get(PendingTray);
end;
rewrite(PendingTray);
reset(TempPendingTray);
while not eof(TempPendingTray) do
begin (*copy back 'PendingTray' and count contents*)
  PendingTray := TempPendingTray;
  put(PendingTray);
  get(TempPendingTray);
n := n+1
end;
rewrite(TempPendingTray);
repeat
  writeln;
  repeat
    write('Do you wish to APPEND, to CHANGE, ');
    writeln('to use the SPECIAL facility, ');
    write('or to FINISH .... ');
    InlChar(HainOption)
  until HainOption in ['A', 'a', 'C', 'c', 'S', 's', 'P', 'p', 'F', 'f']
  (*HainOption= S is a special facility,
  used for loading from 'scratch' to 'PendingTray'*)
case HainOption of
  'A', 'a': (* TO APPEND *)
  begin
    writeln;
    repeat
      write('Do you need help [YES or NO] .... ');
      InlChar(HelpOption)
    until HelpOption in ['Y', 'y', 'N', 'n'];
    if HelpOption in ['Y', 'y')
  end;
then
writeln;
writeln('NOTES.');writeln('
(a) Authors and keywords separated');writeln('
(b) To remove the automatic conversion to ");writeln('
('upper case letters');writeln(" begin a line of text with"');writeln('
(a) backslash "\".);writeln('
(c) Date must be a single integer number');writeln('
(d) If addresses are to be entered use the two');writeln('
(t) lines);writeln(" close pack but indicate new'');writeln('
(' lines with a backslash "\".);writeln('
(e) A personal local storage reference');writeln(" may be kept on the 2nd. location line");writeln(" but should be enclosed in square brackets'');writeln(" for example: [BM360]."');end;
repeat
writeln;
writeln('New item:-
writeln;
for a:-1 to 'do
writeln(' I')writeln;
with entry do
begin
writeln( 'Line of author names, or name of addressee : ');
VDUinString(authors);
writeln('First line of title or address : ');
VDUinString(title1);
writeln('Second line of title or address : ');
VDUinString(title2);
writeln('First line of reference location : ');
VDUinString(place1);
writeln('Second line of reference location : ');
VDUinString(place2);
writeln('Date - just the year - : ');
InlInt(date);
writeln('First line of keywords : ');
VDUinString(key1);
writeln('Second line of keywords : ');
VDUinString(key2);
end;
writeln;
OutRecord(entry,n);
repeat
writeln;
repeat
writeln("Do you wish to make a change [YES or NO] .... "
InChar(ChangeOption)
until ChangeOption in ['Y','y','N','n']
if ChangeOption in ['Y','y']
then
change(entry,n)
until ChangeOption in ['N','n']
if entry.date <> NonDate
then
begin
TagEntry.tag := HiTag;
TagEntry.entry := entry;
PendingTray := TagEntry;
pull(PendingTray);
end;
end;
writeln;
repeat
writeln;
repeat
writeln("Do you wish to append more items [YES or NO] .... "
InChar(AppendOption)
until AppendOption in ['Y','y','N','n']
until AppendOption in ['N','n']
end;
C',c'; (* TO CHANGE *)
begin;
writeln;
repeat
writeln("Do you need help [YES or NO] .... "
InChar(HelpOption)
until HelpOption in ['Y','y','N','n']
if HelpOption in ['Y','y']
then
begin
writeln;
writeln("You MUST know the ITEM NUMBERS of the ITEMS you wish to change.'
writeln("If you do not, leave this program and run "bibout" to find them.'
writeln("Changes do not take place immediately, they stay in the PENDING"
writeln("'tay until the "update" program is run.'
writeln("'If an ITEM is changed more than once only the last version survives.'
end;
repeat
10: writeln;
chge := false;
writeln("Type 0 if no ITEM needs changing, otherwise type'
write("the ITEM number... '
InlInt(nn)
if nn>0
then
begin
writeln;
writeln('No negative numbered ITEMS');
end;
if nn > 0 then
begin
writeln;
GetReference(nn);
if not eof(bank) then
begin
entry := bank;
repeat
writeln;
repeat
write('Do you wish to change this item (YES or NO) .... '); InChar(ChangeOption)
until ChangeOption in ['Y', 'y', 'N', 'n'] if ChangeOption in ['Y', 'y'] then
begin
change(entry, nn, chge := true
end until ChangeOption in ['N', 'n']
TagEntry.tag := nn;
TagEntry.entry := entry;
if chge then
begin
PendingTray := TagEntry;
poly(PendingTray, n := n + 1
if not eof(scratch) then
get(scratch);
end;
rewrite(scratch)
end;
(* of Change option *)

'S', 's': (* To move from text file 'scratch' to 'PendingTray' *)
begin
writeln;
write('This option moves the contents of the '); writeln('SCRATCH file into the PENDING tray.');
write('It can be used to copy selected ITEMS from one'; writeln('bibliography to another.');
write('OR, it can be used to reinstate ITEMS'); writeln('which have been changed by the editor.'); writeln;
repeat
writeln;
write('Do you wish these items to be APPENDED, REINSTATED or NO ACTION .... '); InChar(SpecialOption)
until SpecialOption in ['A', 'a', 'R', 'r', 'N', 'n']; if SpecialOption in ['A', 'a', 'R', 'r'] then
begin
reset(scratch);
writeln;
(* now check that scratch holds ITEMS in the correct form *)
if (not eof(scratch)) and
ScratchHoldItems then
begin
while not eof(scratch) do
begin
with entry do
begin
ScratchInStr(authors);
ScratchInStr(title);
ScratchInStr(title2);
ScratchInStr(place);
ScratchInStr(place2);
read(scratch, date);
repeat
read(scratch, ch) until ch = ':';
readln(scratch, TagEntry.tag);
writeln(n, ' Dated ', date, ' Item number ', TagEntry.tag);
ScratchInStr(key1);
ScratchInStr(key2);
end;
end;
end;
end;
end; rewrite(scratch)
end;
end; (* of Special option *)

'F', 'f': begin
writeln;
writeln('Number of ITEMS now in Pending Tray = ', n - 1;
writeln
end; (* of case "MainOption" *)
until HainOption in ['F', 'f']
end. (* end of program Bibin.p *)
program Bibout(input, output, bank, dict, scratch, dlist, PendingTray);
(* To call down items from the bibliography *)
(* written by Tony Heyes, Blind Nobility Research Unit,
Department of Psychology, The University,
Nottingham, U.K. *)

label 10;

const LineLn = 70;
RowLn = 20;
HitTag = 10000;
LinesPerPage = 64;
VDULinesPerPage = 24;

var FileAssigned1 boolean, bank, pendingTray1 file of item,
dlist, AddressFile, scratch1 text,
dict: file of dict,
FirstLink, SecondLink, ThirdLink, pl, here1 link,
low, high, n, NumSoFar,
LineNo, AddLineNo, count, TopItem, NFromDict, NumW: integer;
Device, FileStyle, MainOpt, NDOption, LogicAction: char;

var ch: char;
a, OrdZero: integer;
NegFound: boolean;

procedure InlChar(var ch: char);
(* to read the first character of a word typed into the terminal *)
Begin
ch := input;
while not (ch in ['A'..'Z', 'a'..'z']) do
begin
(* skips along until first character found *)
get(input);
if eoln(input) then
begin

writeln;
write('ERROR: character required .... ')
end;
ch := input;
end;
while not eoln(input) do (* skips over rest of line *)
get(input);
end; (* of InlChar *)

procedure InlInt(var f: text; var int: integer);
(* to read an integer and not cause a fatal error if a character is given
character required *)

var ch: char;
a, OrdZero : integer;
NegFound : boolean;

begin
repeat (* skips along until integer is found *)
get(f);
if eoln(f) then
begin
writeln;
write('ERROR: digit required .... ')
end;
ch := input;
until (ch in ['0'..'9'])

if ch='+' then
begin
NegFound := true;
get(f);
ch := input;
end
else
begin
NegFound := false;
if ch='-' then
begin
get(f);
ch := input;
end
end;
a := 0;
OrdZero : ord('0');
repeat
a := 10*a+ord(ch)-OrdZero;
get(f);
ch := input;
until not (ch in ['0'..'9'])

while not eoln(f) do (* skips over rest of line *)
get(f);
if NegFound then


int := -a
else
int := a
end; (* of IntInt *)

procedure SkipToEndOfPage(PageLines : integer;
- var where : text);
begin
while LineNo < PageLines do
begin
writeln(where);
LineNo := LineNo+1
end;
LineNo := 0
end; (* of SkipToEndOfPage *)

procedure GetRef(n : integer; destination : char);
var a,CharCount,LineInQuestion,NofCommas,WordLength : integer;
line : string;
DoubleSpace,InBrackets,KeepNextCap,
something,KeepAllCaps,woops : boolean;
ch,LastCh : char;
begin
if n<count
then begin
reset(bank);
count := 1
end;
while (count < n) and (not eof(bank)) do
begin
count := count+1;
get(bank)
end;
if eof(bank)
then begin
writeln;
writeln('You have only got ',count-1,' Items. ');
writeln;
goto 10
end
else
begin
case destination of
'T','t': (* Output to terminal *)
begin
if (VDULinesPerPage-LineNo < 9)
then
SkipToEndOfPage(VDULinesPerPage,output);
for a:=1 to 7 do
write('-------I');
begin
  woops := false;
  writeln(AddressFile);
  AddLineNo := AddLineNo +1;
  write(AddressFile, ' ')
end
else
  write(AddressFile, ch);
  DoubleSpace := (ch-'') and (LastCh-''),
  LastCh := ch
end;
end;
while (AddLineNo mod 8) <> 0 do
  writeln(AddressFile),
  AddLineNo := AddLineNo + 1
end;
if woops
then
  writeln;
  writeln;
  write('All attempts to output a reference');
  writeln('in address format.');
  writeln,
  writeln,
  rewrite(scratch); FileAssigned := false,
  goto 10
end
begin (* firstly the author line *)
  writeln(scratch, '.nr'),
  (* this is an NROFF macro *)
  write(scratch, '\:'); (* bold lettering command *)
  DoubleSpace := false;
  KeepAllCaps := false;
  woops := false;
  LastCh := 'I'; (* initial value *)
  CharCount := 0;
  NOFCommas := 0;
  if authors[1]=''
  then
    begin
      KeepAllCaps := true;
      CharCount := CharCount +1
    end;
    while (CharCount<LineLn) and not DoubleSpace do
      begin
      CharCount := CharCount+1;
      ch := authors[CharCount];
      if ch=''
      then
        NOFCommas := NOFCommas+1;
        DoubleSpace := (ch=' ') and (LastCh=' ');
        LastCh := ch
      end;
      DoubleSpace := false;
      LastCh := 'I';
      CharCount := 0;
      while (CharCount<LineLn) and not DoubleSpace do
        Begin
        CharCount := CharCount+1;
        ch := authors[CharCount];
        if ch=''
        then
          NOFCommas := NOFCommas+1;
        DoubleSpace := (ch=' ') and (LastCh=' ');
        LastCh := ch
        end;
        DoubleSpace := false;
        LastCh := 'I';
        CharCount := 0;
        while (CharCount<LineLn) and not DoubleSpace do
then
  InBrackets := true;
  if InBrackets
    then
      if line[CharCount]=''
        then
          begin
            line[CharCount] := '';
            InBrackets := false
          end;
      end;
  if InBrackets
    then
      line[CharCount] := ''
    until CharCount=LineLn
end; (* of case LineInQuestion *)
CharCount := LineLn;
repeat
  CharCount := CharCount-1
until (CharCount=1) or (line[CharCount]=' '); if CharCount<LineLn
then
  line[CharCount+1] := ''; (* a silly character *;
  (* placed at the end of the character string *)
WordLength := 0;
if CharCount>1
then
  repeat
    CharCount := CharCount-1;
    if line[CharCount]=''
      then
        begin
          if line[CharCount] in ['A'..'Z']
            then
              WordLength := WordLength+1
          WordLength := 0
        end
      else
        begin
          if not (WordLength in [2,3])
            then
              line[CharCount] := '-'
            (* another silly char fills up spaces
              before words which keep caps. *)
          WordLength := 0
        end
  until CharCount=1;
CharCount := 0;
something := false;
if line[1]=''
then
  begin
    KeepAllCaps := true;
    CharCount := CharCount+1
  end;
ch := ':'; (* initial value *)
while (CharCount < LineLn) and
  (line(CharCount) <> '!') do
begin
  CharCount := CharCount+1;
  LastCh := ch;
  ch := line[CharCount];
  if not ((LastCh in ['.','']) and
      (ch in [' ','!']))
    then
      begin
        if (ch in ['A'..'Z']) and not KeepNextCap
          then
            ch := chr(ord(ch)+32));
        if ch in ['A'..'Z']
          then
            KeepNextCap := false;
        if ch="\'
          then
            woops := true; (* its an address *)
        if ch="'
          then
            begin
              ch := '';
              if (LineInQuestion in [3,4])
                then
                  KeepNextCap := true
            end;
        if ch in ['1'..'9'])
          then
            KeepNextCap := false;
        if (ch<>')' and (ch<>')')
          then
            if something := true;
            if something
              then
                write(scratch,ch)
        end;
if something
  then
    writeln(scratch)
end,
if woops
  then
    begin
      writeln;
      writeln;
      writeln('An attempt to output addresses in');
      writeln('reference format.');
      writeln;
      writeln;
      rewrite(AddressFile);
      FileAssigned := false;
end;
goto 10
end
end (* of 'R' *)
end (* of case destination *)
end

procedure Rewind(var ptr : link);
var p,q,pt: link;
begin
p := ptr;
pt := nil;
while p<>nil do
begin
new(q);
q^.val := p^.val;
q^.next := pt;
pt := q;
here := p;
p := p^.next;
dispose(here)
end;
ptr := pt
end; (* of Rewind *)

procedure GetDict(m: integer, var ptr: link);
var a: integer;
p: link;
OldEntry: dic;
more: boolean;
begin
if m < H1Tag then
begin
reset(dict);
a := 1;
while a<m do
begin
OldEntry := dict^;
get(dict);
if OldEntry.cont=false then
a := a+1
end;
writeln;
writeln(dict^.
for a:=1 to RowLdo
if dict^.
then
begin

procedure join(var pl: link, p2: link, which: char);
var continue: boolean;
q,qp,pt1,pt2,pt3: link;
begin
pt1 := pl;
pt2 := p2;
continue := (pt1<>nil) and (pt2<>nil);
qp := nil;
case which of
'A','a': (* AND *)
begin
while continue do
begin
if pt1^.val>pt2^.val then
begin
pt3 := pt1;
pt1 := pt2;
pt2 := pt3;
end;
if pt2^.val>pt1^.val then
begin
pt1 := pt1^.next;
continue := pt1<>nil
end
else if pt1^.val=pt2^.val then
begin
end;
end;
new(q);
q:.val := pt1:.val;
q:.next := qp;
qp := q;
pt1 := pt1:.next;
pt2 := pt2:.next;
continue := (pt1<>nil) and (pt2<>nil)
end
end; (* of AND *)

'O','o': (* OR *)

begin
begin
while continue do
begin
if pt1:.val>pt2:.val
then
begin
pt3 := pt1;
pt1 := pt2;
pt2 := pt3
end;

if pt1:.val<pt2:.val
then
begin
new(q);
q:.val := pt1:.val;
q:.next := qp;
qp := q;
pt1 := pt1:.next;
continue := pt1<>nil
end
else
if pt1:.val=pt2:.val
then
begin
new(q);
q:.val := pt1:.val;
q:.next := qp;
qp := q;
pt1 := pt1:.next;
pt2 := pt2:.next;
continue := (pt1<>nil) and (pt2<>nil)
end
end

if pt1=nil
then
pt1 := pt2;
while pt1<>nil do
begin
new(q);
q:.val := pt1:.val;
q:.next := qp;
qp := q;
end;

end;

(* of case *)

if pt1=nil
then
begin
new(q);
q:.val := pt1:.val;
q:.next := qp;
qp := q;
end;

(* of join *)

procedure OutList(ptr : link; var aa : integer);

var p : link;
begin
p := ptr;
aa := 0;
end;
write(p'.val;5)
end;
writeln;
writeln
end; (* of OutList *)

procedure DictList(var where : text);
(* TO LIST DICTIONARY *)

const NoOfLines = 64;
WordsPerLine = 4; (* Change constants to suit page size *)
(* See also line 700 *)
type list = array[1..384] of word;
var num,i : integer;
OldEntry : dic;
WordList : list;
begin
reset(dict);
rewrite(dlist);
i := 0;
while not eof(dict) do
begin
for num=1 to NoOfLines*WordsPerLine do
begin
OldEntry := dict;
while (dict'.cont=true)and(not eof(dict)) do
get(dict);
if not eof(dict)
then
begin
WordList[num] := OldEntry.name;
end
end;
begin
for num=1 to NoOfLines do
begin
write(WordList[num],WordList[NoOfLines+num],
get(dict));
end;
end; (* of reverse *)

begin
reset(F);
if not eof(F)
then
begin
page(G);
writeln;
writeln('Output in two column "Xerox" label format.');
writeln
end;
while not eof(F) do
procedure GetFromDict(var FirstWord, NumWords: integer);

var
  ch, action, option : char;
  n, ChCount, PointerNum, NumberFound : integer;
  name, signame : word;
  AllCaps : boolean;

begin
  writeln;
  AllCaps := true;
  ChCount := 0;
  write('Enter word required or [.] .... ');
  repeat
    read(ch);
    until ch=''
  if ch='
    then
      begin
        AllCaps := false;
        read(ch);
      end;
    if ch='.'
      then
        begin (* "action" *)
          while not eoln(input) do
            get(input);
            repeat
              writeln;
              writeln('Do you wish to LOOK UP the selected string, to [.] .... ');
              writeln('selection or to QUIT the dictionary: .... ');
              InChar(action);
            until action in ['L', 'l', 'R', 'r', 'O', 'o', 'Q', 'q']
          end else begin (*word*)
            action := 'W';
            repeat
              ChCount := ChCount + 1;
              if ChCount > 1
                then
                  read(ch);
                if AllCaps and (ch in ['a'..'z'])
                  then
                    name[ChCount] := chr(ord(ch)-32)
                  else
                    name[ChCount] := ch
                until eoln(input) or (ChCount = 20);
              if not eoln(input)
                then
                  readln;
              for n := ChCount+1 to 20 do
                name[n] := ' ';
            end;
        end;
      end;
    end;
  end;
end;(* release(here) *)
then
FirstWord := -1 (* look up *)
else
if action in ['R','r']
then
FirstWord := -2 (* restart *)
else
if action in ['Q','q']
then
FirstWord := 0 (* quit *)
else
if name="***
(* special word *)
then
begin
writeln;
writeln('*** ALL ITEMS ***');
writeln;
repeat
write('Is this correct [YES or NO] .... ');
InChar(option)
until option in ['Y','y','N','n']
if option in ['Y','y']
then
FirstWord := HiTag
else
GetFromDict(FirstWord,NumWords)
end
else
begin (* a real word *)
reset(dict);
NumberFound := 0;
PointerNum := 0;
writeln;
signame := ' ';
while (name := signame) and not eof(dict) do
begin
if name=signame
then
begin
writeln(dict".name);
NumberFound := NumberFound+1
end;
while (dict".cont=true) do
get(dict);
if (PointerNum > 0) and not eof(dict)
then
get(dict);
PointerNum := PointerNum+1;
for n=1 to ChCount do
signamen[n] := dict".name[n];
for mr=ChCount+1 to 20 do
signamen[n] := ' ';
end;
else
begin
writeln;
if NumberFound=0
then
begin
writeln('Word not found in your dictionary; try again.' );
writeln;
GetFromDict(FirstWord,NumWords)
end
else
begin
repeat
if NumberFound = 1
then
write('Is this word correct [YES or NO] .... ')
else
write('Are ALL these words required [YES or NO] .... ')
InChar(option)
until option in ['Y','y','N','n']
if option in ['Y','y']
then
FirstWord := PointerNum - NumberFound;
NumWords := NumberFound
end
else
GetFromDict(FirstWord,NumWords)
end
end; (* of GetFromDict *)

begin (* MAIN PROGRAM *)
rewrite(scratch);
rewrite(AddressFile);
reset(bank);
count := HiTag;
LineNo := 0;
AddLineNo := 0;
FileAssigned := false;
writeln;
writeln('To retrieve ITEMS from the BIBLIOGRAPHY. ');
(* TO SEARCH BY AUTHORS and KEYWORDS *)
writeln;
reset(dlist);
if dlist".= i="
then
InInt(dlist,TopItem)
else
begin
TopItem := 0;
writeln('Counting, please wait. ');
writeln;
writeln;
while not eof(bank) do
begin
TopItem := TopItem + 1;
get(bank)
end;
rewrite(dlist);
write(dlist,'-',TopItem : 5);
write(dlist);
write(dlist,'Your DICTIONARY must first be compiled by running');
write(dlist,'the HARD COPY option of 'bibout'.');
write(dlist);
write(dlist);
writeln('The BIBLIOGRAPHY currently holds ',TopItem,' ITEMS.');
repeat
writeln('Do you wish to obtain a HARDCOPY of the current dictionary,');
write('to SEARCH for items or to FINISH .... '); InlChar(MainOpt) until MainOpt in ['H','h','S','s','F','f',
writeln;'output to TERMINAL or to scratch FILE .... '); InlChar(device) until device in ['T','t','F','f','S','s',
writeln;'New sequence.');
repeat
write('Is the desired output');
write('an ITEM list,'); write('the full item being given');
write('a REFERENCE list,'); write('only the reference part being given');
write('an address list suitable'); write('for ENVELOPE addressing .... '); InlChar(FileStyle)
FileAssigned := true until FileStyle in ['I','i','R','r','E','e'); if FileStyle in ['R','r')
begin
writeln('Words are looked up in '); writeln('the dictionary and a list of reference numbers'); writeln( 'containing the given word is shown on the terminal.'); writeln; write( 'The special "word", [***] will match with all the words'); writeln( 'in the dictionary.'); writeln; write('Logical combination of '); writeln('author and keywords continue until you wish'); writeln('to terminated the search.'); writeln('To terminate a search answer the prompt with a full stop [.].');
repeat
write Bien;
NumSoFar := 0; (* mark [here] *)
GetFromDict(NFromDict,NumW);
if NFromDict > 0 (* a real word *) then
begin
GetDict(NFromDict,FirstLink);
if NumW > 1 then
repeat
NFromDict := NFromDict + 1;
GetDict(NFromDict,SecondLink);
joiFirstLink,SecondLink,'O');
```
NumW := NumW - 1
until NumW = 1;
OutList(FirstLink, NumSoFar);
while NF romDict > 0 do
begin
GetFromDict(NFromDict, NumW);
if NFromDict > 0 (* a real word *)
then
begin
GetDict(NFromDict, SecondLink);
if NumW > 1
then
repeat
NFromDict := NFromDict + 1;
GetDict(NFromDict, ThirdLink);
NumW := NumW - 1
until NumW = 1;
OutList(SecondLink, NumSoFar);
repeat
write('AND, OR or NOT ..... ?
');
InChar(LogicAction)
until LogicAction in ['A', 'a', 'O', 'o',
'N', 'n'];
join(SecondLink, ThirdLink, LogicAction)
OutList(FirstLink, NumSoFar)
end;
end;
if ((NumSoFar > 0) and (NFromDict = -1))
then (* look up *)
begin
write;
write('Search in progress for', NumSoFar, ' Items');
write;
pt1 := FirstLink;
while pt1<>nil do
begin
GetRef(pt1'.val, FileStyle);
here := pt1;
pt1 := pt1'.next;
dispose(here)
end;
if FileStyle in ['I', 'i', 'R', 'r', 'E', 'e']
then
begin
write;
write('ITEMS written to SCRATCH FILE.'
);
write
end;
(* release(here) *)
end;
end;
until NF romDict=0 (* quit *)
end;
```
if FileStyle in ['R', 'r']
then
begin
writeln('The output file "scratch" contains the references and the instructions for the word processing program "nroff".');
writeln('An attempt has been made to reintroduce lower case letters.');
writeln('To obtain your output run "nroff scratch"');
writeln('If all is not well edit scratch and run "nroff scratch" again.');
writeln('When all is correct get the hard copy output by running "nroff scratch lpr"');
end;
if FileStyle in ['E', 'e']
then
TwoCols(AddressFile, scratch);
end. (* of program Bibout.p *)
integer;

procedure FromCore;

var p : link;
begin
writeln(scratch,'FromCore');
rewrite(dict);
GotFromCore := true;
P := first;
while p<>nil do
begin
    dict := pA.dline;
    put(dict);
    here := p;
p := pA.next;
    dispose(here)
end
end;(*OfFromCore*)

procedure build(entry : item;n:integer);
(*TO BUILD THE DICTIONARY*)

var boolean;
 entry.authors;entry.key1;
 entry.key2

begin
for line:=l+1 to 3 do
begin
    case line of
        1: str := entry.authors;
        2: str := entry.key1;
        3: str := entry.key2
    end;
    l := 0;
    let := 0;
then
    begin
        repeat (* not empty line *)
            let := let+1;
            LastWord := (((str[let]=' ') and (str[let+1]=' '))
                or (let=Lw-1));
            WordFound := ((str[let]=' ') or LastWord);
        if not WordFound
        then
            begin
                l := l+1;
                if (l=1) and (str[let]=' ') then
                    l := 0
                else
                    begin
                        if l<2 then
                            NewEntry.name[l] := str[let]

                        end;
                    end;
            end;
        end;
    end;
end;

for i:=1+1 to 20 do
NewEntry.name[i] := ' ';
(* fill up with spaces *)
if InitialBuild
then
begin (* first entry *)
    NewEntry.numbers[1] := n;
    for i:=2 to RowLn do
        NewEntry.numbers[i] := 0;
    NewEntry.cont := false;
    new(p);
pA.dline := NewEntry;
pA.next := nil;
first := p;
l := 0;
InitialBuild := false
end
else begin
    OldEntry := firstA.dline;
    pt := first;
    (* move pt past all words before the new entry *)
    while (ptA.next<>nil) and
        (NewEntry.name<>ptA.nextA.dline.name) do
        pt := ptA.next;
    OldEntry := ptA.dline;
    same := OldEntry.name=NewEntry.name;
    space := OldEntry.numbers[rowLn]=0;
    AlreadyHad := false;
    if same then
    begin
        i := RowLn;
        while OldEntry.numbers[i] = 0 do
            i := i-1;
        if OldEntry.numbers[i] = n then
            AlreadyHad := true
        end;
    if not AlreadyHad then
    begin (* if keyword has author name only one dic *)
        if (same and (not space))
        then
            begin
                (* new entry already in dict but no space in the string *)
                OldEntry.cont := true;
                ptA.dline := OldEntry
            end;
        if same and space
        then
            begin
                (* new entry already in dict AND space in the number string *)

            end;
        end;
    end;
end;

for i:=2 do
begin
    NewEntry.numbers[i] := n;
end
end;(*OfBuild*)

end;(*OfBibupdate*)
(* a new word for the dictionary OR a repeat of an old word *)
NewEntry.number := n;
NewEntry.cont := false;
for i := 2 to RowLn do
  NewEntry.number[i] := 0;
new(newp);
newp.dline := NewEntry;
if NewEntry.name < first.dline.name then
  begin (* new head of the list *)
    newp.next := first;
    first := newp;
  end else
  begin (* slot entry into list *)
    newp.next := pt.next;
    pt.next := newp
  end end;
(* of AlreadyHad *)
I := 0
end until LastWord
end; (* of build *)

procedure merge; (* to merge dict in core with existing dict on file *)
var continue : boolean;
j, jj: integer;
NewEntry : dict;
begin
  writeln(scratch, ' Merge');
  rewrite(TempDict);
  reset(dict);
  (* copy to scratch with additions *)
  pt := first;
  continue := (not eof(dict)) and (pt.next<>nil);
  while continue do
    begin
      if dict.name<pt.dline.name then
        begin
          put(TempDict);
          get(dict);
          continue := not eof(dict)
        end
      else if dict.name=pt.dline.name then
        begin
          TempDict := pt.dline;
          put(TempDict);
          here := pt;
          pt := pt.next;
          dispose (here);
          continue := pt<>nil
        end
      if dict.dline.name<pt.dline.name then
        begin
          dict.cont := true;
          TempDict := dict;
          put(TempDict);
          get(dict);
          continue := not eof(dict)
        end end;
  while pt<>nil do
    begin
      TempDict := pt.dline;
      put(TempDict);
      here := pt;
      pt := pt.next;
      dispose (here)
    end; rewrite(dict);
    reset(TempDict);
    (* copy back to dict and squeeze *)
    while not eof(TempDict) do
      begin
        NewEntry := TempDict;
        if (NewEntry.number[RowLn]>0) or (NewEntry.cont=false) then
          begin
            dict := NewEntry;
            put(dict);
            get(TempDict)
          end
        else
begin
  get(TempDict);
  if not eof(TempDict) then
    begin
      for j:=2 to RowLnd do
        if NewEntry.numbers[j] = TempDict^numbers[1] then
          begin
            NewEntry.numbers[j] := TempDict^numbers[1];
            for jj:=1 to RowLnd-1 do
              TempDict^numbers[jj] := TempDict^numbers[jj+1];
            TempDict^numbers[RowLnd] := 0;
          end;
        end;
    end;
  rewrite(TempDict)
end; (* of merge *)

begin (* MAIN PROGRAM *)
reset(PendingTray);
reset(bank);
dlistOK := false;
rewrite(Scratch);
writeln(Scratch, 'To deal with corrections');
writeln(Scratch, 'Corrections ', corr:5, ' Additions ', addiS);
while (not eof(correction)) and (n<stack) do
  begin
    n := n+1;
    new(e);
    e^TagEntry := correction;
    e^next := nil;
    efirst := e;
    get(correction);
    while (not eof(PendingTray)) and (not distOK) do
      begin
        if dist^next = nil then distOK := true;
        add := 0;
        Topltem := 0;
        reset(dlist);
        if dist = '-' then distOK := true;
        rewrite(PendingTray)
        begin
          if not distOK then
            begin
              while not eof(bank) do
                begin
                end;
            end;
        end;
    end;
end; (* order correction into core in batches of 'stack' *)
writeln(Scratch, 'To deal with corrections');
writeln(Scratch, 'Corrections ', corr:5, ' Additions ', addiS);
while (not eof(correction)) do
  begin
    n := n+1;
    new(enewp);
    enewp^TagEntry := correction;
    if correction^tag<efirst^tag then
      begin
        (* new head of list *)
        enewp^next := efirst;
        efirst := enewp
      end;
    else
      begin
        (* move pointer ept to correct place, slot in new item *)
        distOK := false;
        rewrite(PendingTray)
      end;
  end;
end;
ept := efirst;
while (ept^.next<>nil) and (correction^.tag=ept^.TagEntry.tag)
do
  ept := ept^.next;
  if correction^.tag=ept^.TagEntry.tag
  then
    ept^.TagEntry := correction
  (* replace with later correction, this is why items are sorted in this way *)
  else
    begin
      enewp^.next := ept^.next;
      ept^.next := enewp
    end
end,
get(correction)
end; (* n=stack or eof(correction) *)
write(scratch,'Corrections processed in ');
writeln(scratch,'this batch', n:5);
(* first batch of items from 'correction' now in core and ordered *)
(* now read bank to TempBank making changes from core. Items are labelled for later extraction by making the date = NonDate. Replacement items are passed to join additions. *)
write(scratch,'Copy bank to TempBank ....');
rewrite(TempBank);
reset(bank);
OldTotal := 0;
ept := efirst;
while not eof(bank) do
begin
  OldTotal := OldTotal+1;
  if (ept^.next<>nil) and (ept^.TagEntry.tag=OldTotal)
  then (* we have found one to correct *)
    begin
      if ept^.TagEntry.entry.date<>NonDate
      then (* ie. it is not empty *)
        begin
          (* Replacement item written to addition file *)
          write(addition,ept^.TagEntry.entry);
          reps := reps+1
        end;
      bank^.date := NonDate;
      write(TempBank,bank^);
      get(bank);
    end;
(* Making the date = NonDate will remove the item when the last batch of corrections are processed *)
  now := ept;
  ept := ept^.next;
  dispose(now)
end
else
begin
  write(TempBank,bank^);
end;
get(bank)
end
(* read TempBank back to bank *)
write(scratch,'Copy TempBank to bank ....');
rewrite(bank);
reset(TempBank);
while not eof(TempBank) do
if eof(correction) and (TempBank^.date=NonDate)
  then
    get(TempBank) (* removes corrected items *)
  else
    begin
      write(bank,TempBank^);
      get(TempBank);
    end; (* of reading back to bank *)
write(scratch,' O.K.');
rewrite(TempBank);
end; (* return for more corrections *)
rewrite(correction);
reset(addition);
while not eof(addition) do
begin
(* order additions alphabetically into core in batches of 'stack' *)
write(scratch,'To deal with additions.');
if reps>0
  then
    writeln(scratch,'These include ',reps:5,
    ' replacements.');
(* mark(now) *)
n := 1;
nnew(e);
e^.TagEntry.entry := addition^;
e^.next := nil;
efirst := e;
get(addition);
while not eof(addition) and (n<stack) or
begin
  n := n+1;
nnew(enewp);
enewp^.TagEntry.entry := addition^;
move := ((enewp^.TagEntry.entry.authors
  > efirst^.TagEntry.entry.authors) or
  (enewp^.TagEntry.entry.authors
  = efirst^.TagEntry.entry.authors) and
  (enewp^.TagEntry.entry.date
  > efirst^.TagEntry.entry.date)));
if not move
  then (* new head of list *)
    begin

begin
 write(TempBank,ept^.TagEntry.entry);
 now := ept;
 ept := ept^.next;
 dispose (now)
end;
 LastOne := bank;
(* assigned to give LastOne a starting value *)
 writeln(scratch,'O.K.');

(* now copy back to bank *)
 write(scratch,'Copy TempBank to bank ...');
 reset(TempBank);
 rewrite(bank);
(* release(now) *)
while not eof(TempBank) do
 begin

same := ((TempBank^.authors=LastOne.authors)
          and (TempBank^.title1=LastOne.title1)
          and (TempBank^.title2=LastOne.title2)
          and (TempBank^.date=LastOne.date));
if not same
 then
 write(bank,TempBank^);
(* rejects duplicates
 LastOne := TempBank^;
 get(TempBank)
end;
 writeln(scratch,'O.K.');
 rewrite(TempBank)
end; (* return for more additions *)
end; (* of dependence on PendingTray *)

(* TO BUILD THE DICTIONARY *)
reset(bank);
reset(dict);
rewrite(addition);
rewrite(PendingTray);
if eof(dict) then
 begin
 n := 0;
 InitialBuild := true;
 m := 0;
(* mark(here) *)
 writeln(scratch,'To build dictionary');
while not eof(bank) do
 begin
 n := n+1;
 m := m+1;
 build(bank^,n);
 get(bank);
 ii := heap
 then
...
begin
if not GotFromCore
  then
  FromCore
else
  (* release (here) *)
  (* mark (here) *)
  InitialBuild := true;
  n := 0
end;
if not GotFromCore
then
  FromCore
else
  merge;
  (* release (here) *)
end;
if n > 0 then TopItem := n;
if not listOK then
begin
  rewrite (list);
  writeln (list, ': TopItem := 5);
  writeln (list);
  write (list, 'DICTIONARY must be compiled by running ');
  writeln (list, 'the HARD COPY option of 'bibout!');
  writeln (list);
end
end. (* of program Bibupdate.p *)

Dear Nick

After our phone conversation the other week, I was rather relieved to feel that here in the UK there are other Pascalers at work and that PASCAL 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 contribute with public domain software, but at the moment, I don't have a great deal of time to spare, nor any telecomm equipment to plug into my computer.

I enclose a cheque for 9 pounds for subscription. On the question of hack 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 sending your time collating everyone's needs and doing individual photocopies of hits and pieces. Perhaps if other people were able to land you some of the older copies, the same could be done. I'd certainly lend you 12-16 if you like. After all, it's the information that matters, not whether the issue is an original or not unless we have any 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 as course as Wordstar to compose programs and write letters. The hardware is OEM kit from Sinclair computers in Purley, by the name of Midlands and is an Integrand 10-slot S100 case with S100, Ithaca IEEE 5100 cards (MPU-80, FDC-2, 64K DR and 100 boards) giving 64K and 4MB Z80A with CP/M, plus Z80E-DATA 1740 1MB drives. The printer is a Rane (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 novelties. I have written to Charles Foster of Pascal/P User Group asking if he or his contributors would permit the distribution of any of their Pascal sources to PUGUK members appropriately modified to PS 6197, or if indeed there is any other Public Pascal around in the States. I think we ought to be prepared to recirculate this, don't you?

In converting from programming mainly on mainframes in Fortran and having a modding 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 PASCAL 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 newsletter 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 RS 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.

Anyway, the best of luck

John R Logsdon

Tongue-in-cheek Pascal Language enhancements.

a) Structured constants.

Program make-up to be for example:

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

exptype= RECORD
  a: integer;
  b, c: char;
  d: array[0..3] of integer;
  f: scalar type;
  g: set of scalar type;
  h: array[1..20] of char
END;

.................. etc

TABLE exl:exptype=
  onehundred, 'a', chr(20), (0, 25, 50, 75), jam,
  [coffee, tea, bread], 'cholesterol';

VAR exvar: exptype; display1: char;

BEGIN
  exvar:= exl;
  display1:= 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. This 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 corresponding 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 genuine structured read-only constants without the ugly initation presently necessary in Pascal. In fact, in practice it is easier to put records for initialisation in a parameter file and read them in, which does not seem an elegant solution. For micros with restricted memory, initialising 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 AARC Pascal 8000 for the 144 160/370 series, and attributed to Hugheasmus)

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

---

Name: Mr P A E Herring

Address: MAPAC

17 Market Square

Leighton Buzzard

Bedfordshire

LU7 7EU

Phone: 0525 378 237

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 interm ISO standard or UCSD for Burroughs B1955 with 0.5 M Byte working store. Manufacturer does not support Pascal for.

---

P.S.1. Can you recommend a Pascal for XENIX

P.S.2. Do you know who distributes the Dutch Free University version of APL809? (in the UK)

P.S.3. Please send a receipt

P.S.4. Please note change of address!

P.S.5. Thanks.
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 licence and send it together with your remittance to:

Dr Z J Ciechanowicz
Division of Information Technology & Computing
National Physical Laboratory
Teddington
Middlesex TW11 0LW 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.

Yours sincerely

Z J CIECHANOWICZ

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

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

COUNCIL FOR EDUCATIONAL TECHNOLOGY FOR THE UNITED KINGDOM
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 ES 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 based upon the test suite together with other material.

The test suite consists of about 17300 lines of Pascal programs plus additional 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.

An application form for a licence to use the suite is on the other side of this notice.

E.A. Wichmann,
Z.J. Ciechanowicz, extension 3977,
For ESI, J. Hatton-Smooker, telephone 0442 3111
APPLICATION FOR LICENCE 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 activities. In such documents, reference shall be made to the original copyright notice and the source.

Signed

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

Date

FINIS