UNIVERSITY OF EDINBURGH
Edinburgh Regional Computing Centre

User Note 64

(September 1986)

Title:

Pascal on EMAS-3

Author: Ken Chisholm

Contact: Advisory service

Software Support Category: See Note 15

Synopsis

This User Note describes the use of the Pascal compiler as implemented on EMAS-3. It assumes that the reader is reasonably conversant with the Pascal language for which there are many good introductory text books (see the Appendix.) It mainly concentrates on the ways in which this Pascal extends the ISO standard specification, which is the British Standard and is fully described in the BSI document BS 6192: 1982, Computer Programming Language Pascal.

Keywords

Pascal

Edinburgh Regional Computing Centre

James Clerk Maxwell Building, The King's Buildings, Mayfield Road, Edinburgh, EH9 3JZ. Telephone 031-667 1081

1 Introduction

The new Pascal compiler on EMAS-3 is based on one originally developed at UMIST by Atholl Hay and Jim Welsh and has been further developed and implemented for EMAS by the ERCC Compiler Group. It supports an extended dialect of the language called ICL Pascal, but can enforce strict conformance to the ISO Standard by means of a compiler PARM option. This Note describes how to use the compiler on EMAS-3 and provides a brief introduction to the ICL Pascal extensions. The new compiler also differs from the current Southampton implementation by providing conformant array parameters, full set of char, and enforcing all compile-time checks required by the ISO Standard.

2 Calling the Compiler

The compiler is called using the command Pascal. It takes four parameters of which the first is obligatory:

```
Command: pascal source, object, listing, errors
```

The source file should be a character file containing a Pascal program; it can be more than one file joined to another with '+', e.g.

```
decls+procs+prog
```

The `object` file is used to hold the compiled program. If a file of the name given does not already exist it will be created. If an object file of the given name does exist then its contents will be overwritten. If this parameter is omitted, the file T#OBJECT is created (or overwritten).

The `listing` parameter can be the name of a file or a device for the compiler listing. If it is omitted, a default listing file called T#LIST is used. The user may subsequently examine this file by using, for example, LOOK or SHOW, or listing the file on a line printer, e.g.

```
Command: list t#list, .lpnn
```

The `errors` parameter can be the name of a file or device to receive the compile time error messages: the default is .OUT.

As with other compilers on EMAS-3 the PARM command can be used before calling the compiler to set various compiler options. The Parm options available with Pascal are:

Nolist	- no source lines placed on listing.
Nowarnings	- no warnings, only errors reported.
Nocheck	- no unassigned variable checking or de-reference checking (when implemented).
Noarray	- no array bound or range checking.
Opt	- nocheck, noarray and no line numbers.
Strict	- no extensions allowed to ISO standard.
Notrace	- no diagnostic information in the compiled programs (implies Nodiag and Noline).
Nodiag	- diagnostics restricted to a traceback of procedures.
Noline	- no line numbers recalled in traceback or failure messages.
R8	- reals allocated 8 bytes of storage instead of the default of 4 bytes.

3 Porting Pascal programs to use Pascal on EMAS-3

It is thought that existing Pascal programs on EMAS-2900 should transfer to the Pascal compiler with relatively few changes. Some of these changes are mentioned or discussed in the next sections.

As with Fortran-77 and IMP80 on EMAS-3, the EMAS command calling mechanism will be standardized to use the external procedure EMAS3 - e.g.

```
program EMASExample;

procedure EMAS3(readonly Command : packed array[l1..u1: integer] of char;
                readonly Parms : packed array [l2..u2: integer] of char;
                var Flags : integer);
   extern; emas;

var
   Flags: integer;

begin
   ...
   EMAS3('Define', 'l,.lpl5', Flags);
   ...
end.
```

A list of the procedures comprising the Subsystem interface on EMAS-3 is given in User Note 80. Thus for prompting, Pascal, as with other languages, will use the standard external procedure EMAS3PROMPT:

```
procedure EMAS3Prompt(readonly p : packed array[l..u: integer] of char);
   extern; emas;
```

Since the parameter is a conformant array, the string can be of arbitrary length. A call will be of the form:

```
EMAS3PROMPT('No. of Rows?: ')
```

3.1 Error Action

A full list of Pascal compile time error messages may be found in the file SUBSYS:PASCALERRS1.

A run time error reporting and diagnostic system, similar to that used by the other compilers on EMAS-3, has been implemented for use with the Pascal compiler.

4 Language Extensions

EMAS-3 Pascal is very similar to the ICL Pascal language which is a true superset of the ISO Standard for the language, BS 6192 level 1. There are also a number of local, EMAS-3 specific extensions (such as the EMAS-3 command calling procedure). Briefly, the essential differences from the ISO standard Pascal are:

- Some lexical extensions
- Separate compilation
- Initialisation of variables at declaration
- Relaxation of order of declarations
- Default (otherwise) clause for case statements
- Default variants in record types
- Various type extensions
- Mixed-language programming.

These differences are described in more detail in the following sections.

4.1 Lexical Extensions

(i) A character string may be continued over more than one line, using the ampersand character '&' as illustrated below:

```
'This string ' &
'occupies 4 lines ' &
' and may include ' & { as shown here }
' embedded comments'
```

(ii) The symbol '|' may be used as an alternative to '&'.

(iii) Identifiers may contain break characters '_'. These are useful to separate words and so make the names easier to read, e.g.

```
three_word_identifier
```

(iv) The symbol '->' may be used as an alternative to '^'.

(v) The symbol '\' may be used in a character string or character constant to include a control character, thus:

\b	for backspace
\t	for horizontal tab
\n	for newline
\f	for form feed
\r	for carriage return
\ddd	for the character whose value is given by the octal digit sequence ddd
\\	for backslash itself.

4.2 Separate Compilation

Pascal software can be written in a modular fashion using the keywords **visible** and **extern** which allow data and procedures to be linked between modules by name. For example, the variable f and the procedure newpage in the program below:

```
program m(f);

var
   f : text; visible;

procedure newpage; extern;

begin { program }
   rewrite(f);
      ...
   newpage;
      ...
end.
```

```
program n;

var
   f : text; extern;

preset
   pageno: integer:=0;

procedure newpage; visible;
begin
   pageno:=succ(pageno);
   page(f);
   writeln(f, 'page ', pageno:5)
end;

begin
end.
```

4.3 Initialised variables (preset, readonly)

Pascal provides two forms of initialised variables:
`preset` variables are initialised when the software is prepared for execution, that is, before any block has been activated. Apart from this initialisation, `preset` variables behave exactly as variables declared in the normal variable declaration part of the program.
`readonly` variables have fixed values (equivalent to %const variables in IMP80). The semantics of the language does not permit these variables to be used in contexts where their values could be changed.

For example:

```
type
   r = record
      a : packed array [ 1..4 ] of char;
      b : integer;
      c : Boolean
   end;

preset
   v : r := ( a =>'FRED', b => 2, c => false );

readonly
   q : array [ 1..2, 3..4] of integer := ((0, 1), (1, 0));
```

The structure and values of the initialisation are determined by the type of the variable being initialised, as illustrated in the following sections.

4.4 Simple Variable Initialisation

```
const
   smallint = 255;
   space = ' ';

type
   weekend = (Saturday, Sunday);
   byterange = 0..smallint;
```

```
readonly
   pubnight : weekend := Saturday;
   halfbyte : byterange := 127;
   compilertrace : Boolean := false; visible;
   filler : char := space;
   trigger : char := ' ';
   options :integer; extern;
   error_bound : real := 0.01;
```

Note that visible and extern may be used with initialised variables as in the example above.

4.5 Set Initialisation

```
type
   day = ( Monday, Tuesday, Wednesday, Thursday,
           Friday, Saturday, Sunday);
   colour = ( red, orange, yellow, green, blue,
              indigo, violet);
   days = set of day;
   hue = set of colour;

readonly
   weekdays : days := [Monday..Friday];
   sickly : hue := [green, red..yellow, violet];
   initdays : days := [];
```

4.6 Positional Array Initialisation

```
type
   days = (Monday, Tuesday, Wednesday, Thursday,
           Friday, Saturday, Sunday);
   months = (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug,
             Sep, Oct, Nov, Dec);
   friends = (Andy, Keith, Pat, Ben, Mark);
   height = array [1..2] of integer;

readonly
   daynames : array [days] of packed array [1..3] of char
              :=('Mon','Tue','Wed','Thu','Fri','Sat','Sun');
   daysinmonth : array [months] of integer
                 := (31,28,31,30,31,30,31,31,30,31,30,31);
   heights : array [friends] of height
             := ((5,11),(5,8),(5,0),(3,4),(3,1));
   onezeros : array [1..5] of integer := (1, otherwise=>0);
   allzeros : array [1..4] of integer := (otherwise=>0);
```

In the above example, `daysinmonth[Feb]` will have the value 28 and `heights[Ben,2]` will have the value 4.

4.7 Indexed Array Initialisation

Example 1

```
preset
   a : array [1..10] of integer := (1..3 & 5 & 6 & 8..10 => 0,
                                    4 & 7 => 1);
   b : array [1..10] of integer := (4|7 =>1, otherwise => 0);
```

Example 2

```
type
   months = (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug,
                        Sep, Oct, Nov, Dec);
preset
fouryears : array [1..4] of array [months] of 0..31 :=
   (1|2|3 => (Feb => 28, Apr|Jun|Sep|Nov => 30, otherwise => 31),
   4 => (Feb => 29, Apr|Jun|Sep|Nov => 30, otherwise => 31));
```

4.8 Relaxed Ordering of Declarations

The strict ordering of declarations enforced in ISO Standard Pascal has two connected disadvantages. First, it can prevent logically related groups of declarations from being textually related. Second, it makes the inclusion of external declarations difficult. Like many other Pascal compilers, EMAS-3 Pascal supports relaxed ordering of declarations. For example

```
program OrderExample;

var
      seed : integer; extern;

function random(max : integer): integer; extern;

const
   nullparam = -1;

type
   access = (read, write);
```

4.9 Mixed-Language Programming

It is possible to call IMP80 and Fortran-77 routines and functions by specifying them to be external procedures as specified above in Section 4.2. The following table indicates the correspondence between the Pascal formal parameters and the IMP80 and Fortran-77 formal parameters which are valid between the languages.

Pascal	IMP80	Fortran-77
INTEGER value	%INTEGER	no equivalence
INTEGER variable	%INTEGERNAME	INTEGER
REAL value	%REAL	no equivalence
REAL variable	%REALNAME	REAL
REAL variable	%LONGREALNAME	DOUBLE PRECISION (R8 option in force.)
BOOLEAN value	%INTEGER	no equivalence
BOOLEAN variable	%INTEGERNAME	LOGICAL
anything else	no equivalence	no equivalence

In the case of functions, it is also necessary to ensure that the function result corresponds to a suitable type. The following table shows the correspondence between valid function types.

Pascal	IMP80	Fortran-77
INTEGER	%INTEGER	INTEGER
REAL	%REAL	REAL
REAL	%LONGREAL	DOUBLE PRECISION (R8 option in force.)
BOOLEAN	%INTEGER	LOGICAL

Notes

1. PASCAL booleans appear to IMP as integers taking the value 0 for FALSE and 1 for TRUE.

2. Procedures can only be passed to another language if all the parameters of the procedure can be represented in the other language.

3. Expressions must be assigned to a local variable before being passed to Fortran-77 - there is no call by value.

4.10 OTHERWISE Clause for CASE Statements

As in many other Pascal compilers, the default "catch-all" **OTHERWISE** clause is permitted when using the **CASE** statement. For example:

```
var ch : char;
   n : integer;
      ...
case ch of
'I' : n := 1;
'V' : n := 3;
'X' : n := 8;
otherwise n := 0
end
```

4.11 Default Variant in Record Types

In an analogous manner to the otherwise clause in case statements, the otherwise keyword may be used in record types to associate a variant with remaining values of the tag-type. This is shown in the example below. It can also be used in initialisation statements to set all other unspecified parts of arrays to some value. Examples of this are shown in Section 4.3 on "Initialised Variables".

```
type
   Form = record
      case Kind: integer of
         1: (r: real);
         2: (b: Boolean);
         3: (c: char);
      otherwise
            (i: integer)
      end;
```

4.12 Type Extension (word)

A predefined type called word is available to declare variables using various number bases. For example

```
var i,j,k : word;
   .....
   .....
begin
   i := 2#101; { This sets i to equal 5}
   j := 8#77;
   k := 16#1FF
end.
```

5 Acknowledgements

This User Note is based on earlier documentation written by Rosemary Soutar, John Blair-Fish and Roderick McLeod and also several discussions with Atholl Hay and Rob Pooley of the ERCC.

Appendix A: Pascal Bibliography

A concise Pascal bibliography with suitable annotations

With the growing interest in the programming language Pascal in and out of the University, a subset (by no means exhaustive) of recommended publications is outlined below, both for the novice and the sophisticated user.

1. Wilson, I.R., and Addyman, A.M., A Practical Introduction to Pascal - with BS6192. Macmillan, 1982.
   Well written, with carefully chosen examples. Comes easily in the category of a "best buy". Includes the text of the ISO Standard definition of Pascal.

2. Welsh, J., and Elder, J., Introduction to PASCAL, Second Edition, Prentice Hall, 1983.
   Written with superb clarity and includes several case studies where the style of programming goes hand in hand with current methods of structured programming and stepwise refinement. This second edition conforms to the ISO standard Pascal definition.

3. Grogono, P., Programming in PASCAL, Addison-Wesley, (Second edition).
   Comprehensive coverage of the language, suitable for those who can already program.

4. Findlay, W., and Watt, D.A., An Introduction to Methodical Programming, Pitman, 1978.
   Good examples, and comes recommended as "a comprehensive and thoughtful treatment which combines teaching of PASCAL with the principles of program construction".

5. Rohl, J.S. and Barrett, H.J., Programming via Pascal, Cambridge University Press, 1980
   A good textbook around which a lecture course might be constructed
   Excellent syntax diagrams.

6. Jensen, K. and Wirth, N., PASCAL User Manual and Report, Third edition, 1985, Springer-Verlag.
   A revised edition of the original user manual and report, now updated to align with the ISO Standard.