

ISO PASCAL



ACORN
SCIENTIFIC
REFERENCE
MANUAL



ISO PASCAL



PART NO 0410, 010
ISSUE NO 1
JULY 1985

© Copyright Acorn Computers Limited 1985

Neither the whole or any part of the information contained in, or the product described in, this manual may be reproduced in any material form except with the prior written approval of Acorn Computers Limited (Acorn Computers).

The product described in this manual and products for use with it, are subject to continuous developments and improvement. All information of a technical nature and particulars of the product and its use (including the information in this manual) are given by Acorn Computers in good faith.

In case of difficulty please contact your supplier. Deficiencies in software and documentation should be notified in writing, using the Acorn Scientific Fault Report Form to the following address:

Sales Department
Scientific Division
Acorn Computers Ltd
Fulbourn Road
Cherry Hinton
Cambridge
CB1 4JN

All maintenance and service on the product must be carried out by Acorn Computers' authorised agents. Acorn Computers can accept no liability whatsoever for any loss or damage caused by service or maintenance by unauthorised personnel. This manual is intended only to assist the reader in the use of the product, and therefore Acorn Computers shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any error or omission in, this manual, or any incorrect use of the product.

Published by Acorn Computers Limited,
Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN.

Within this publication the term BBC is used as an abbreviation for the British Broadcasting Corporation.

NOTE: A User Registration Card is supplied with the hardware. It is in your interest to complete and return the card. Please notify Acorn Scientific at the above address if this card is missing.

ISBN 0 907876 38 2 Acorn Scientific

Contents

1	Introduction to ISO Pascal	1
1.1	The Pascal language	1
1.2	Using ISO Pascal	1
1.2.1	Installation	2
2	Using the compiler	3
3	A Description of ISO Pascal	7
3.1	Language reference	7
3.2	Restrictions and variations	7
3.2.1	Implementation-defined features	7
3.2.2	Implementation-dependent features	9
3.2.3	Data sizes	10
3.3	Extensions to ISO 7185	11
3.3.1	Identifiers	11
3.3.2	Non-decimal constants	11
3.3.3	Bit-vector operators	11
3.3.4	Reset and rewrite	11
3.3.5	Miscellaneous functions	12
3.3.6	OTHERWISE	12
3.3.7	\$INCLUDE	12
3.3.8	Machine-code	13
3.3.9	Modules	13
4	Compatibility	17
4.1	Compatibility with Acornsoft ISO Pascal	17
4.1.1	Implementation-defined features in Acornsoft Pascal	17
4.2	Data sizes in Acornsoft Pascal	18
4.3	Limitations in Acornsoft Pascal	18
4.3.1	Identifiers	18
4.3.2	Non-decimal constants	19
4.3.3	Bit-vector operators	19
4.3.4	Miscellaneous functions	19
4.3.5	OTHERWISE	19
4.3.6	INCLUDE	19
4.3.7	Machine-code	19
4.3.8	Modules	20
4.3.9	Warning messages	20
4.4	Extensions in Acornsoft Pascal	20

5	Errors and debugging	21
5.1	Compile-time messages	21
5.1.1	Warning messages	21
5.1.2	Non-Fatal Error messages	26
5.1.3	Fatal compile errors	29
5.2	Run-time errors	29
6	Using Pascal with other languages	31
6.1	Introduction	31
6.2	Conformance	32
6.2.2	Types	32
6.2.3	Parameters	36
6.3	Interfacing to Panos standard procedures	39
6.4	Examples	41
	Appendix A	47
	Appendix B	99
	Appendix C	103

1 Introduction to ISO Pascal

1.1 The Pascal language

The Pascal programming language is a versatile, well-structured, and strongly-typed high level language originally developed for use as a teaching language, but now used for a wide variety of system and applications software.

The major features of Pascal are its ease of use, legibility, rigorous checking during compilation to reduce the number of coding errors and source portability.

Note that this document is a reference manual intended as a guide to the Acorn 32000 implementation of ISO Pascal; it is not a tutorial. Throughout this document, ISO Pascal refers to Pascal as implemented on Acorn 32000 based products and running under the Panos Operating System.

1.2 Using ISO Pascal

ISO Pascal is a compiled language. It consists of a two-pass compiler which translates Pascal source programs into 32000 machine code, and a collection of pre-compiled modules (the Pascal library) to provide facilities such as string to numeric conversion.

The first pass of the compiler checks that the program conforms to the rules of Pascal and the second pass constructs the equivalent machine code program.

In addition, useful facilities such as the ability to call routines written in other languages (such as C and FORTRAN 77) have been included. This uses the Acorn cross-calling standard introduced for use with Pascal in chapter 6 of this document and described in full in the *Panos Technical Reference Manual*.

The Pascal compiler produces extensive debugging and error trapping information, though this can be disabled using compiler options. There are also various extensions to the standard, e.g. bit-wise logical operations.

1.2.1 Installation

The Pascal compiler is supplied on a 5¼ inch double-sided floppy disc. Procedures for installation are shown in the relevant chapter in the appropriate *User Guide* supplied with this system. If the correct procedures are not followed, then Pascal may not function correctly. Note that even though Pascal has been supplied on floppy disc, and is going to be used with the DFS, it still must be installed. If you wish to know more about file extensions or utilities in general, consult the *Panos Guide to Operations*.

2 Using the compiler

There are a number of arguments which can be issued to the compiler which give extra control over the input and output of the compilation, and allow the compilation options to be used.

-list

This command specifies that a line-numbered listing is sent to 'Prog-lis' (by default), or a given file name.

For example: Pascal PProg -list Pprogn will place the listing in Pprogn-lis. The listing consists of the original source program along with numbered lines, and any messages that may have been generated during compilation. This can provide a useful reference when de-bugging. See figure 1 for a demonstration of this command. The numbering scheme is very simple: each physical line in a source file is numbered sequentially starting from one. Some line numbers are followed by the characters 'l' or '+'. These constitute aids to tracking down potential sources of errors.

The symbol 'l' signifies that the line is a continuation of a comment started on a previous line. Omitting the closing right brace '}' from a comment is a common mistake resulting in subsequent statements being taken as part of the comment. An 'extended comment' will be terminated by the next properly specified comment, leading to parts of the program disappearing mysteriously.

The '+' character indicates that the line is a continuation of the previous line, and pin-points places where semicolons might have been omitted.

-aof

The file containing the machine code equivalent program in Acorn Object Format, is automatically sent to a file with the same name as that of the source file, but with the extension '-aof'. This command allows the user to specify the name of the Acorn Object Format file. For example: Pascal Prog -aof ObjProg will place the object program in a file called 'ObjProg-aof'. Unless an alternative extension is given, the default '-aof' is added by the compiler.

-extend

This option makes the compiler accept programs containing any of the extensions to standard Pascal. By default, the compiler will not

recognise any of the extensions and use of them will result in compile-time error messages.

-nowarn

This option prevents the compiler from issuing warning messages. As warning messages can often indicate trouble spots in a program, this option should only be used by programmers experienced in the use of the compiler.

-notify

If the '-extend' option is used and '-notify' is also given, then

```
Warning 158 -- this is a non-standard construction
```

is issued each time an extension is found.

-nochecks

This option should only be used with programs that have already been fully tested. It prevents the compiler from generating code to

- i. detect the use of unassigned variables while the program is executing;
- ii. detect overflow during the evaluation of expressions;
- iii. check that enough memory is available for the correct execution of the program.

-noDiags

This option inhibits the generation of, i.e. information used in the event of program failure which locates the faulty source line by number and displays the names and values of active variables. Other than the behaviour following execution errors, this option has no effect on the execution of a program; however, it does reduce the size of the object file.

-identify

This option requests identification of the compiler.

-help

This option prints out help text.

Example compiler commands

A. The Minimal Command

Pascal Prog1

This command will compile the source program 'Prog1-Pas'; default settings are used throughout, i.e. the aof file is called 'Prog1-aof', no listing is generated, error messages are sent to the screen, warning messages are given, full checking and diagnostic tables are included in the object code, and Pascal extensions are unacceptable.

B. Specifying an error file and inhibiting diagnostic tables

```
Pascal dfs::2.Prog1 -error dfs::0.errs -nodiags
```

Compilation is carried out as in the first example, except that error messages are sent to the file 'dfs::0.errs', and no diagnostic tables are included in the object code.

一、二、三、四、五、六、七、八、九、十、十一、十二、十三、十四、十五、十六、十七、十八、十九、二十、二十一、二十二、二十三、二十四、二十五、二十六、二十七、二十八、二十九、三十、三十一、三十二、三十三、三十四、三十五、三十六、三十七、三十八、三十九、四十、四十一、四十二、四十三、四十四、四十五、四十六、四十七、四十八、四十九、五十、五十一、五十二、五十三、五十四、五十五、五十六、五十七、五十八、五十九、六十、六十一、六十二、六十三、六十四、六十五、六十六、六十七、六十八、六十九、七十、七十一、七十二、七十三、七十四、七十五、七十六、七十七、七十八、七十九、八十、八十一、八十二、八十三、八十四、八十五、八十六、八十七、八十八、八十九、九十、九十一、九十二、九十三、九十四、九十五、九十六、九十七、九十八、九十九、一百

3 A Description of ISO Pascal

3.1 Language reference

The language reference followed for ISO Pascal is the ISO 7185 Standard, available in a slightly revised form as BS 6192:1982 *Specification for Computer programming language Pascal*, from the British Standards Institution, and reprinted elsewhere.

3.2 Restrictions and variations

3.2.1 Implementation-defined features

This section describes those details of the language implementation which are left as implementation-defined (“possibly differing between processors but defined for any particular processor”), by the ISO 7185 Standard. The numbers in brackets below refer to sections in that standard.

(6.1.7)

String-characters may be any ASCII character. NL (ASCII 10) is currently excluded pending clarification of various statements made in the standard validation suite.

(6.4.2.2-b)

The real-type corresponds to the 64-bit double-precision floating-point numbers of the proposed IEEE standard for Binary Floating Point Arithmetic (Task P754).

(6.4.2.2-d)

The char-type defines values corresponding to the ASCII 7-bit character set, extended to include the ordinal values 128 .. 255.

(6.6.5.2)

Except in the case of text files the operations `rewrite`, `put`, `reset` and `get` are performed at the time the corresponding statement is executed. For text files the operation is delayed until the next actual or implied use of the buffer variable associated with the file.

(6.7.2.2)

MAXINT has the value 2147483647 (16_7FFFFFFF).

(6.7.2.2)

Real operations are performed to the accuracy of the proposed IEEE standard for Binary Floating Point Arithmetic (Task P754). Floating-point functions return results correct to 12 decimal places.

(6.9.3.1)

The default values for TotalWidth are:

Integer-type : 1

Real-type : 3

Boolean-type : 5

(6.9.3.4.1)

ExpDigits has the value 3.

(6.9.3.4.1)

The exponent character is 'e'.

(6.9.3.5)

The value TRUE is output as 'True' and the value FALSE is output as 'False', in other words for both values all letters are in lower case except for the first which is in upper case.

(6.9.5)

PAGE causes a form-feed character (ASCII 12) to be sent to the specified file.

(6.10)

Program parameters may only denote files, and these may only be of types which do not include pointers. Program parameters are bound to external files which correspond to the newline-delimited text files of the host system.

(6.1.9)

The following table gives the alternative tokens accepted by the compiler:

Reference token	Alternative token
-----------------	-------------------

↑	\
[(
]	.)
{	(*
}	*)

3.2.2 Implementation-dependent features

This section describes those details of the language implementation which are left as implementation-dependent (“possibly differing between processors and not necessarily defined for any particular processor”) by the ISO 7185 Standard. The numbers in brackets below refer to sections in that standard.

(6.5.3.2)

The index-expressions of indexed variables are evaluated from left to right.

(6.7.1)

The expressions of a member designator are evaluated from left to right.

(6.7.1)

The member-designators of a set constructor are evaluated from left to right.

(6.7.2.1)

The operands of a dyadic operator are evaluated from left to right.

(6.7.3)

The actual parameters of a function-designator are evaluated from left to right.

(6.8.2.2)

The variable of an assignment statement is accessed before the expression is evaluated.

(6.7.3)

The actual parameters of a procedure-statement are evaluated from left to right.

(6.9.5)

Inspecting a textfile to which PAGE was applied will return the ASCII character Form File (code 12). This does not mark the end of a line.

(6.10)

Program parameters may only be bound to external entities which are files.

(6.6.5.2)

The file parameters to READ and WRITE are evaluated the number of times given by:

MAX(1), number of parameters involving function calls

(6.6.5.4)

The array parameters to PACK and UNPACK are evaluated once each for each implied assignment.

3.2.3 Data sizes

Integer	4 bytes, doubleword aligned
Real	8 bytes, doubleword aligned
Char	1 byte, byte aligned
Boolean	1 byte, word aligned
Set	32 bytes, doubleword aligned
Records	n bytes, doubleword aligned
Arrays	(first element), doubleword aligned
Enumerated	
1 .. 255 items	1 byte, byte aligned
256 .. 32767 items	2 bytes, word aligned
otherwise	4 bytes, doubleword aligned
Subrange (Ordinal value)	
0 .. 255	1 byte, byte aligned
-32768 .. 32767	2 bytes, word aligned
otherwise	4 bytes, doubleword aligned

3.3 Extensions to ISO 7185

Important: these extensions (with respect to ISO 7185) will only be accepted if the compiler is run with the `-extend` option.

3.3.1 Identifiers

Dollar (\$) and underline (_) may be used as the second or subsequent characters in identifiers.

3.3.2 Non-decimal constants

The form `BASE_DIGITS` may be used to specify constants with radix other than ten. `BASE` is the decimal specification of the radix, and `DIGITS` is a sequence of letters or decimal digits where the letters `A`, `B`, `C` represent the values 10, 11, 12 For example:

$$16_CAFE3 = 831459 = 8_3127743$$

3.3.3 Bit-vector operators

The following operators act on fullword integer values:

<code>&</code>	logical AND
<code> </code>	inclusive OR
<code> </code>	exclusive OR
<code><<</code>	logical left shift
<code>>></code>	logical right shift
<code>~</code>	one's complement

The precedence of these operators is as follows:

<code>&</code> , <code><<</code> , <code>>></code>	same as <code>*</code>
<code> </code> , <code> </code>	same as <code>+</code>
<code>~</code>	same as unary <code>-</code>

3.3.4 Reset and rewrite

The file procedures 'reset' and 'rewrite' are extended to enable file variables to be linked with external (i.e. filing system) names. The first parameter is

the same as in the standard procedures. The second parameter is a string giving the name to be used by the current filing system. An example is:

```
reset (infile, ':1.text1');
rewrite (outfile, 'text2');
```

The strings may be variables, e.g.

```
reset (data,userName);
```

A special case of the extended forms for 'reset' and 'rewrite' is the null string parameter, or carriage-return, e.g. 'reset (file, '')'. In this case, the file is bound to the console i.e. the screen for output and the keyboard for input. The file must be a text file. This is useful in enabling users to specify the console as a file, for example, so that text can be seen on the screen instead of being sent to a disc file which has to be examined later. Programs which process data until the 'eof' condition becomes true for a file can still work when the keyboard is used. The character **CTRL** - **D** used as input sets eof to true for the input file (or any file reset as the console).

3.3.5 Miscellaneous functions

LINENUMBER returns the number of the source line containing the reference to the function. Source lines are numbered starting from one for each file contributing to the compilation (see **\$INCLUDE**).

ADDRESS(anyvar : anytype) returns the machine address of the variable given a parameter.

SIZE(anyvar : anytype) returns the number of bytes occupied by the variable given as parameter.

3.3.6 OTHERWISE

The reserved word **OTHERWISE** may be used to introduce the final clause of a **CASE** statement making that clause the action to be taken if no other clause in the case statement has been selected.

3.3.7 \$INCLUDE

The **\$INCLUDE** statement may be used to include source text from other files during the compilation. The form of the statement is:

```
$INCLUDE 'a-file-name'
```

The lines in each included file are numbered from one.

3.3.8 Machine-code

The compiler permits a form of assembly language to be included in program text. It should be noted at the outset that this facility is only intended as a last resort when other methods of achieving a result are totally inappropriate. No responsibility whatever is accepted for erroneous behaviour of programs containing machine-code statements.

The general form of the machine-code statement is:

```
*<mnemonic>_<operand list>;
```

where <mnemonic> is a 32000 standard assembly language instruction and <operand list> is a list of one or more operands separated by commas. For example:

```
*MOVD_1,X;
*SVC_12;
*ADDR_142(0),4;
```

Because Pascal uses integer values rather than identifiers for labels, destinations of branch instructions must be followed by a colon:

```
*BNE_99;;
```

Machine-code does not permit access to non-local variables.

The following examples demonstrate the various operand types:

```
*MOVD_#123456,2;
*MOVD_1,%external(1)+4;
*MOVD_X,%TOS;
*MOVD_12(7)[4:%B],0(%SB);
*MOVD_12(16(%PC)), 16(12(%FP))
```

3.3.9 Modules

Modules provide a means for separate compilation. A module may contain definitions of procedures, functions and variables, some of which may be

made available to other programs or modules by preceding their definition with the reserved word **EXPORT**. In order to overcome limitations in linkage mechanisms and to give flexibility to naming conventions, an optional **ALIAS** may be given to a procedure or function identifier. This alias defines the string to be used for external linkage; it has no naming significance within the text of the program. If no alias is given the procedure or function name is used as the linkage name.

Procedures and functions which have been exported from other modules may be referenced by giving a procedure or function heading preceded by the reserved word **IMPORT**.

Variables global to the module must be prefixed by one of **EXPORT**, **IMPORT** or **STATIC**. The use of **VAR** for global variable declarations is not allowed in modules. Preceding variable declarations by **EXPORT** allows them to be accessed by other modules that contain similar variable declarations preceded by **IMPORT**.

Global variables preceded by **STATIC** remain local to that module but retain their values between calls to that module. There exists a mechanism to allow **STATIC** and **EXPORT** variables to have initial values assigned at their declaration.

For example:

```
TYPE colour = (red, green, blue);
EXPORT counter : integer := 0;
EXPORT a : ARRAY [1..6] OF integer := 12,7,23,5,1,5;

STATIC stream : integer;
STATIC called : boolean := false;
STATIC colourmap : ARRAY [2..5] OF colour := red, red, green, red;
```

The variable 'counter' is initialised to zero. The integer array 'a' is initialised to the six values following the variable declaration. Arrays cannot be partially initialised, i.e. all or none of the elements should be initialised. The variable 'stream' has no initial value but, once set, will hold its value between calls to this module. The boolean variable 'called' is initialised to the value 'false' and will have this value when the module is first entered. Neither 'stream' nor 'called' can be accessed from outside this module.

The whole module must start with the statement: **MODULE** module-identifier; and end with the statement **END**.

For example:

```
MODULE example;
  EXPORT counter : integer := 0;

  STATIC called : boolean := false;

  IMPORT size : integer;

  IMPORT FUNCTION other ALIAS 'EY_FN'(w : integer) : integer;

  FUNCTION setup(p : integer) : integer; {a local function}
  BEGIN
    counter := counter+1;
    if called then setup := other(counter*size)
      else setup := other(p*size);
    called := true;
  END;

  EXPORT FUNCTION inner ALIAS 'EX_INNER'(p, q : integer) : integer;
  BEGIN
    inner := setup(p) + other(p*q*size);
  END;
END. {of module}
```

In this example 'setup' is only accessible locally whereas 'inner' is available to other modules/programs via the linkage name 'EX_INNER'.

