

C



ACORN
SCIENTIFIC

REFERENCE
MANUAL



C



PART NO 0410, 007
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 40 4 Acorn Scientific

Contents

1	Introduction	1
1.1	Standard C	1
1.2	Installation	1
2	Using the compiler	3
2.1	Compiler Options	3
2.2	Panos Global Variables	5
3	Differences from Standard C	7
3.1	Restrictions	7
3.1.1	Loose type checking of . and -> operators	7
3.1.2	White space within compound operators	7
3.1.3	Use of sizeof in array declarations	7
3.1.4	#line Ignored	8
3.1.5	Anachronisms not allowed	8
3.2	Extensions	8
3.2.1	Dollar sign in identifiers	8
3.2.2	More significant characters in identifiers	8
3.2.3	Assignment to whole struct/union variables	9
3.2.4	Long escape sequences	10
3.2.5	Restrictions on struct member names	10
3.2.6	Type-name syntax relaxed	10
3.2.7	Data Type void	11
3.2.8	Forward References to structure tags	11
3.3	System-dependent features	12
3.3.1	Data Type enum not allowed	12
3.3.2	Pointers to Functions	12
3.3.3	Standard locations for #include files	12
4	The C Runtime Library	13
4.1	The Purpose of the runtime library	13
4.2	Conventions	13
4.3	Library Modules	15
4.3.1	Input/Output Routines	16
4.3.2	Mathematical Functions	21
4.3.3	String Handling	22
4.3.4	Character Classification	23
4.3.5	Conversions	23
4.3.6	Dynamic Memory Allocation	24

4.3.7	Miscellaneous	24
4.4	Alphabetic List of Functions	25
5	Debugging	51
5.1	Compiler error message format	51
5.2	Fixing errors detected by the compiler	54
5.3	Errors detected during execution	56
	Appendix A	57
	Appendix B	61

1 Introduction

32000 C refers to the implementation of C on Acorn Cambridge Series computers, under the Panos operating system. Note that this manual is not a tutorial; reference to such material may be found in Appendix B.

1.1 Standard C

The definition of 32000 C follows that of Kernighan and Ritchie which is described in *The C Programming Language*. Exceptions are noted in chapter 2 below. Throughout this manual, Kernighan and Ritchie's definition is called standard C, although it is not a formal national or international standard. Because most other implementations of C are also based on Kernighan and Ritchie's definition it is possible to move C programs quite freely between different computers, provided that extensions to the standard peculiar to individual machines are avoided. Although much of the power of C comes from the library routines for input and output of data, string handling etc. which are supplied along with most compilers, the standard does not define a set of routines which all compilers must provide. However, most compilers agree about the definitions of the commonly used routines. The library routines supplied with 32000 C (see Chapter 4), with the exception of the low-level I/O routines, are common to almost all implementations of C; it is not guaranteed however that these routines will have exactly the same effect as their counterparts in other versions of C.

1.2 Installation

The installation procedure is described in the User Guide supplied with the system. C is provided on a DFS format disc, and must still be installed even if it is going to be used on this medium.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

2 Using the compiler

Having created the source using an editor, giving it the file extension '-c', the command

```
cc <program name>
```

will compile the program. The compiler automatically searches for files with the '-c' extension, and it does not need to be quoted in the command.

2.1 Compiler Options

A number of compiler options are available which allow greater control over the input and output of the compilation, and over various debugging tools.

The overall argument string is:

```
{-source} name [{-list (name)}] [{-aof (name)}] [-module name]
[-decode] [-nodiags] [-check] [-error name] [-show] [-identify] [-help]
```

where name refers to any file name, and braces enclose optional items. Note that some of the file names will contain extensions. File extensions are an important Panos feature. The extensions '-lis' and '-aof' are automatically appended by the compiler for list files and Acorn Object Format (aof) files respectively. See the *Panos Guide to Operations* for further details about file extensions.

-source

This argument is optional and does not need to be quoted when giving the source name. The source program must always have the file extension '-c' appended to the file name.

-list

This option generates a listing which is sent to the file name specified. The file extension '-lis' is automatically appended. See figure 2 (section 5.1) for an example of an erroneous program compiled from within the editor, where the listing is sent to another file, also loaded into the editor.

-aof

Normally, the Acorn Object Format file which results from the compilation is given the same name as the source, with the extension '-aof' instead of the extension '-c'. This option allows the user to specify a different name. The '-aof' extension is added by the compiler.

-module

The module name field of the generated aof file is normally set to be the root name of the corresponding source program file (with the '-c' extension removed). Use `-module <name>` to override this default and set the module name field in the aof file to `<name>`.

-decode

If this option is used, the aof file generated by the compiler will contain extra information which allows the `-decode` command to produce a listing showing the machine code for each source statement in the program. To interpret this, a program will need to be written.

-nodiags

The code generated by the compiler contains some diagnostic information which allows a backtrace to be produced if an error occurs at run-time. This option disables diagnostic tables. The compiled code is smaller, but error messages are less helpful.

-check

This option causes the compiler to detect when a variable is used before a value has been assigned to it.

-error

This option allows the user to specify a file to which compiler errors are sent. By default, these are sent to the screen.

-show

If this option is used, the listing file will contain the expansion of any macro calls in the source program as well as the original source text. Macro expansion lines are marked by a double quote mark following the line number.

Examples of compiler commands

A. The minimal command

```
cc CProg
```

The source program 'CProg-c' is compiled with all default options: the object file is called 'CProg-aof', and it contains some diagnostic information, but no machine code listing for each source statement; no listing is generated; no checks are made for unassigned variables; errors are sent to the screen.

B. The aof file is specified, and diagnostic tables are disabled.

```
cc -source dfs::3.CProg -aof dfs::1.AFile -nodiags
```

The optional '-source' argument is used to specify 'dfs::3.CProg-c'; the object file is to be called 'dfs::1.AFile-aof', and is to contain no diagnostic information.

C. A listing of the compilation is specified.

```
cc nfs:$.CDir.Cprog -list adfs:$.Cprgl
```

The program 'nfs:\$.CDir.Cprog-c' is compiled, and a listing placed in the file 'adfs:\$.Cprgl-lis'.

2.2 Panos Global Variables

To run C, a number of Panos global string variables must be setup. These are:

cc\$fe	C compiler front end
ll\$be	C compiler back end
C\$include	#include files
Link\$Lib:C	C run-time library
Link\$Lib:Pas	Pascal run-time library
LL\$Prim	Primitive code procedures

These variables must contain the full pathnames of the various files. See the *Panos Guide to Operations* for a description of global string variables.



3 Differences from Standard C

The differences between 32000 C and standard C are described here. Section numbers in the text refer to the section numbers in the *C Reference Manual* (Appendix A of *The C Programming Language*).

3.1 Restrictions

These features of standard C are not permitted in 32000 C.

3.1.1 Loose type checking of . and -> operators

Section 7.1 of the standard states that the left operand of the operators . and -> must be a structure and the right must be the name of a member of that structure. Section 14.1 states however that the compiler allows any value as the left operand of . and any expression of pointer or integer type as the left operand of ->. 32000 C follows the rule of section 7.1 in disallowing examples like the following:

```
int i;
struct { int p, q; } s; /* THIS IS ILLEGAL */

i.p = 0;
i->q = 17;
```

3.1.2 White space within compound operators

In 32000 C, assignment operators like += are single tokens whose parts (+ and =) may not be separated by white space. If + = is written instead of +=, the compiler will produce an error message.

3.1.3 Use of sizeof in array declarations

Constant expressions used in an array declaration may not contain the sizeof operator. This example is ILLEGAL: `char v [sizeof(int)] ;`

3.1.4 #line Ignored

The #line compiler control line is accepted but ignored by the 32000 C compiler.

3.1.5 Anachronisms not allowed

Both of the anachronistic forms ‘=op’ and ‘int x 3;’ described in section 17 of the standard are illegal in 32000 C.

For `x=-1`; write either `x=- 1`; or `x = -1`; depending on which is meant.

For `int x 3`; you must write `int x = 3`;

3.2 Extensions

The following non-standard language features are allowed in 32000 C.

3.2.1 Dollar sign in identifiers

32000 C allows the dollar sign \$ to appear in identifiers. The dollar sign is treated as another letter. The following are all acceptable identifiers:

```
$  
rate$  
$_max9
```

3.2.2 More significant characters in identifiers

Two identifiers are deemed by the compiler to be the same if their first 31 characters match (standard C says 8 characters). Any additional characters are ignored. This rule also applies, while a file of routines is being compiled, to external identifiers, but the rules used to decide whether two external identifiers match when routines compiled separately are linked together depend on the linker program, not the compiler. Most linkers will treat two identifiers as being identical if their first 6 or 7 characters are the same, ignoring upper/lower case distinctions.

If C programs must be portable to many different compilers, they should only use identifiers which are distinct in the first 8 characters, except for external identifiers which should be distinct in the first 6 characters whether or not the distinction between upper and lower case letters is ignored.

3.2.3 Assignment to whole struct/union variables

In standard C, all that can be done with a `struct` variable is to create a pointer to it (using the `&` operator) or access one of its members (using the `.` operator).

32000 C allows you to copy all of the members of a `struct` variable at once by using the assignment operator, `=`. If one operand of `=` is a `struct` then the other must be a `struct` of the same type.

For example:

```
struct { int p, q; } x, y ;
x.p = 3; x.q = 17;
y = x; /* struct assignment */
```

After this structure assignment, `y.p` has the value 3 and `y.q` has the value 17.

Both assignments in the example below are **ILLEGAL** because the types of the operands for `=` do not match.

```
struct { int p, q; } x ;
struct { int a, b; } y ;
int i;

x = i;      /* ERROR one integer, one struct ERROR */
x = y;      /* ERROR same size, but different types ERROR */
```

32000 C also allows function arguments to be `struct` types (standard C allows only pointers to `struct` as arguments). `Struct` arguments are declared and used in the same way as any other type.

For example:

```

struct tag { int p, q; }

clear(x)
struct tag x;
{
    x.p = 0; x.q = 0;
}

example()
{
    struct tag a;

    a.p = 3; a.q = 4;
    clear(a);
    return( a.p + a.q );
}

```

The result returned by the function `example` will be 7 because, like all other types of function argument in C, `struct` arguments are passed by value: `clear` cannot affect the contents of the structure `a` which is passed to it, since it works with a copy of `a` named `x`.

3.2.4 Long escape sequences

Standard C allows octal escape sequences of the form `\ddd` within string and character constants. C2000 does not restrict the length of the digit sequence after the `\` character to three digits, but programs which must be usable with other C compilers should observe the restriction.

3.2.5 Restrictions on struct member names

In standard C, the same member name may occur in different structures only if the fields identified by the member name and all preceding fields are the same. C2000 makes no restrictions on the use of the same member name in different structures. Again, programs which must be usable with other C compilers should not make use of this fact.

3.2.6 Type-name syntax relaxed

Kernighan and Ritchie give the definition of the 'type-name' construct as

```

type-name:
type-specifier abstract-declarator

```

This allows only one 'type-specifier' before the 'abstract-declarator', disallowing expressions like:

```

sizeof(long int)
(unsigned short) e

```

Multiple 'type-specifier's like 'long int' are allowed in this context by other implementations of C, and by 32000 C.

3.2.7 Data Type `void`

The 32000 C compiler has an extra data type `void` which is not part of standard C. `Void` is a special data type with no values, used to indicate that a function returns no value. Expressions can be cast to type `void` in order to discard their value explicitly. The (non-existent) value of a `void` expression may not be used in any way, and neither explicit nor implicit conversions may be applied to such a value. Therefore a `void` expression may be used only as an expression statement, or as the operand of a comma operator.

3.2.8 Forward References to structure tags

Forward references to structure tags are allowed, e.g:

```

struct t1 (int i; struct t2 *p);
struct t2 (char x, y);

```

Using a structure tag (like `t2`) before it has been declared is only allowed when pointers to the structure are being declared or manipulated: objects of that structure can only be declared after the structure has been fully declared. Similarly `sizeof`, `->` and `.` will not work until the complete structure declaration is given.

3.3 System-dependent features

3.3.1 Data Type enum not allowed

32000 C does not allow the data type enum, which is permitted in some C compilers. The enum data type allows the programmer to construct new data types by enumerating the values which variables of that type may take. This restriction is permitted by standard C.

3.3.2 Pointers to Functions

Forward references to static functions are allowed. Static functions are called as external functions, but are defined to the linker as local rather than global symbols, so their names are not exported to other modules.

3.3.3 Standard locations for #include files

If the Panos global string variable C\$include is defined, its value is taken as a list of directories to be searched for #include files, separated by commas or white space. If the form #include "file name" is used instead of #include <file name>, the current directory is searched before the C\$include list. For example:

```
-> set var C$include "$.PanosLib, $.CUser. Fred.CInc"
```

See the *Panos Guide to Operations* for more details about global string variables.

4 The C Runtime Library

4.1 The Purpose of the runtime library

The 32000 C runtime library is a collection of compiled functions which perform commonly-used operations not included in the C language itself: reading and writing data, and evaluation of mathematical functions like `sin` and `cos` are the most obvious instances.

This chapter covers the conventions used to describe the arguments of library routines, lists the available routines grouped by function (I/O, string handling etc.) in section 4.3, and then lists the available routines in alphabetic order, giving a description of the effects of each in section 4.4.

4.2 Conventions

This section describes how to use standard header files in calling library routines and how to interpret the notation used in section 4.4 to specify the number and types of arguments they require.

Runtime library functions are used in exactly the same way as user-defined functions (most are in fact just normal C functions). To use a library function, a program must first declare the name of the function to be used, and indicate that it is external to the program (storage class `extern`).

So that the declarations of library functions in user programs are always correct, standardised header files are provided with the system for each group of library functions. The programmer uses the C `#include` statement to access the contents of the header file before making use of any of the functions declared there.

`#include` files are located by declaring a `Panos` global string variable `C$include` to contain references to these files. The value of the global variable is a list of directories to be searched for `#include` files. The directories are separated by commas or white space. If the form `#include "file name"` is used instead of `#include <file name>`, the current directory is searched before the `C$include` list. For example:

```
-> set var C$include "$.PanosLib, $.CUser. Fred.CInc"
```

See the *Panos Guide to Operations* for more details about global string variables. As well as containing the required function declarations, the header file will include declarations for any special data types required by its functions. For example, consider the standard input/output functions. These are declared in the header file `stdio-h`. Before the first use of any of the standard I/O functions, a program must contain the statement `#include <stdio-h>`. For compatibility with other C compilers, `#include <xxx.h>` is taken as equivalent to `#include <xxx-h>`.

This declares all of the standard I/O functions like `printf` and `getc` as well as defining the macros `EOF` and `NULL` which are used in communication between the I/O functions and user programs. `EOF` has the value `-1`; `NULL` has the value `0`.

Programs should always use the header files provided with the compiler rather than attempting to provide their own declarations for library functions since the declarations of some functions will differ from the obvious declaration implied by the function synopses in this chapter.

These function synopses indicate how to call library functions. Information about required argument types and function result types is presented in the form of a C function declaration prefixed by `#include` statements which indicate which header files, if any, must be used in order to access the function. For example, the synopsis for the `fgets` function looks like this:

```
#include <stdio-h>

char *fgets(s, n, iop)
char *s;
int n;
register FILE *iop;
```

This means that `fgets` returns a result of type `(char *)` and has three arguments of types `(char *)`, `(int)` and `(FILE *)`, where `FILE` is a data type declared in the header file `stdio-h`. This header file must be included in all programs which use the function.

Ellipsis is used in function synopses to indicate that a function has a variable number of arguments, for example the `printf` function:

```
#include <stdio-h>

printf(format[,arg1[,arg2[,...]]])
char *format;
```

The synopsis shows that `printf`'s first argument must be a character pointer. The square brackets `[]` indicate that the enclosed arguments are optional; ellipsis `"..."` indicates repetition. Where argument types are not shown in the synopsis (e.g. `arg1,arg2,...` for `printf`) the allowed argument types are discussed in the text.

4.3 Library Modules

This section lists the library routines provided, divided into the following functional groups.

Stream input/output to files and devices, including facilities for random file access

Classification of ASCII characters (e.g. is a character an ASCII letter?)

String manipulation, including string copy and string comparison

Character conversion (e.g. convert uppercase letters to lowercase)

Numeric conversions between ASCII string and binary representations for integer and floating-point values

Mathematical functions, including logarithms and trigonometric functions

Dynamic ('heap') memory allocation and deallocation

Various other miscellaneous functions

Some background information common to all of the functions in a group is presented in this section rather than being repeated with the description of each individual function. In particular, the concepts on which the standard I/O system is based are presented here, such as `stream`, `file`, `pointer`, etc.