ARNOR C

COMPILER LINKER AND EDITOR

Amstrad PCW8256 PCW8512 CPC6128

Issue 1, 1987

The ARNOR C programs were developed using the MAXAM II assembler and subsequently the ARNOR C compiler.

The editor was developed using the MAXAM II assembler and the ARNOR BCPL compiler.

This manual was written using the PROTEXT word processor and printed from camera-ready copy produced by PROTEXT on a KYOCERA F1010 laser printer. We are indebted to David Foster for his help.

Arnor Ltd., 118 Whitehorse Road, Croydon, CR0 2JF.

CONTENTS

INTRODUCTION

1. INTRODUCTION

a). About the C language

C is undoubtedly the most popular language for commercial programming on microcomputers today. The reasons for this success will soon become apparent.

C is a flexible and efficient language, capable of sufficiently good performance to replace assembly language programming for many tasks. Another feature of the C language is that it is remarkably well standardised, enabling programs to be developed which may then be transferred to other types of computer with the minimum of trouble.

b). The Arnor C compiler

The Arnor C compiler is a full implementation of the C standard as defined by Kernighan and Ritchie in the book "The C programming Language".

The compiler produces compact, fast, intermediate code which runs under an interpreter written in Z80 machine code. The intermediate code is called "Basic Stack Code" and was designed specifically for running C on 8 bit computers. Another advantage of running under the interpreter is that special screen handling routines have been incorporated to allow fast writing to the screen, as well as the use of 'windows'.

This manual describes in detail the operation of the various program modules and everything that is specific to the Arnor C system.

The subject of learning to program in C is extremely large and the manual makes no attempt to teach the language, a subject which is covered comprehensively by many specialist books. Newcomers to C are advised to find a suitable tutorial book.

c). Recommended books

There are dozens of books available, covering the subject of programming in C, from introductory books to those covering specialist aspects of the language. They are written in a variety of styles, from the 'chatty' to the reference book and it is improbable that any one style of book would suit everyone.

The following list contains the names of books which have been found to provide sound and useful information, but it is by no means comprehensive and it is recommended that a visit to a good book shop is worthwhile, in order to select a book with a style that suits you.

Bibliography.

"The C Programming Language" by Brian W. Kernighan and Dennis M. Ritchie (Prentice-Hall, 1978).

This is the definitive guide to the language and, as a result, should be considered an essential work of reference.

"C Programming Guide, 2nd Edition" by Jack Purdum (Que, 1985).

This book is easier reading than Kernighan and Ritchie and provides a clear, comprehensive guide to C.

"C Self Study Guide" by Jack Purdum (Que, 1985).

Suitable for beginners, this book introduces C gradually, with a series of exercises with supplied answers.

"Advanced C Primer++" by Stephen Prata (Sams, 1986).

As the title suggests, this book deals with advanced programming in C. It is well written with many real life examples.

2. ABOUT THE MANUAL

The ARNOR C compiler, editor and associated programs are supplied on a single 3" disc, together with a number of example source files. These, together with this manual, provide all that is necessary to use the C compiler.

IMPORTANT NOTE: With the exception of the first time that the program is used to create a working copy of the disc, the original disc should NEVER be used. It must be retained as a back up and kept in a safe place.

If the original is used and damaged, perhaps by accidentally formatting it, or even spilling a drink on it, you will not have any back up with which to create another working copy.

a). About the manual

The manual is separated into a number of sections, each of which covers one aspect of the Arnor C system. There is no need initially to read the whole manual. In fact there would be far too much information to absorb at one go. Ideally, the sections on 'Getting started' and 'The Editor' should be read thoroughly, as well as the part of the manual concerned with the compiler.

Each section of the manual is split up into chapters, each covering an aspect of the subject and all commands and library functions are covered in detail, with examples where necessary.

Detailed appendices are provided at the back of the manual covering the commands and other special features.

b). Page numbering

Each section of the manual is referred to by name and is further broken down into chapters, which are numbered. Each chapter has its pages numbered, starting at page 1.

Every page of the manual is numbered on the outside top corner of the page and consists of a name and numbers. For example, this section of the manual is the 'Introduction' and this is page 1 of chapter 2. The chapter is the first number and the actual page in the chapter is the number following the hyphen.

c). Version numbers, updates and README

Every Arnor program has a version number. In the case of the compiler and linker, this is displayed when compilation or linking takes place. The run time system displays the version number at the top of the screen when used in interactive mode. The editor displays the version number on the command mode banner line and the individual utility programs display them when they are used (usually at the top of the screen). Any queries regarding the software should be accompanied by the version number of the program concerned and your program registration number must always be quoted.

Arnor have a policy of continual enhancement and improvement of software, and from time to time new versions of Arnor C will be made available. Existing users qualify for a low cost upgrade to any new version, but only if the registration card has been returned to Arnor.

Inevitably the printed documentation cannot always keep up with the changes to the software and so a text file, called 'README', is supplied on the disc, giving any updates to the program or documentation. This file should be loaded into the editor to be read and printed.

3. GETTING STARTED

This chapter explains how to create a 'start of day' disc from the supplied master disc and should now be read and the instructions followed carefully.

WARNING: All the programs and files provided on the master disc are subject to copyright laws and copies may be made for your own use on one machine only. It is an offence to give, hire or sell copies of copyrighted material to other parties.

Arnor C operates under CP/M Plus and is suitable for use on the Amstrad PCW8256, PCW8512 and CPC6128 computers. It may be used with single drive computers, but will take full advantage of two or more drives.

Before Arnor C is used, a working disc, containing copies of the relevant files MUST be made. Make sure that the original disc is 'write protected' before starting to create a 'start of day' disc. If there is any doubt about the use of the write protection tab, full details are given in the Amstrad computer manual.

A 'start of day' disc is one which, when created by following the instructions in this chapter, will contain all the necessary files from the CP/M Plus System Disc and the supplied program disc. It will enable the computer to load CP/M automatically and be ready for use once the disc has been inserted at the start of the day, without the need to insert other discs. CPC6128 users should note that they will still have to type '|CPM' and press RETURN after switching on.

Two newly formatted discs will be required, one for the start of day disc, and one for the compiler system.

a). Creating a 'Start of day' disc

It is assumed that anyone who has bought this program and is intending to write programs in C will already have a basic understanding of the operation of the PCW or CPC computers, so precise details are not given of how to turn on the computer and format discs. Details are given regarding the best way to copy and arrange the various files onto the start of day discs.

The start of day disc should contain the following files, which (except the first two) can be found on side 1 of the supplied master disc.

CP/M system (EMS) file
SUBMIT.COM (from the CP/M system disc)
PROFILE.SUB
STARTUP
APED.COM
RUNC.COM
AC.COM
DCOPY.COM
*.HLP
*.PTR

This disc should be marked 'C Start of Day disc'.

If using a PCW it is recommended that the above files, except the first 4, are copied to the M drive. The file 'STARTUP' contains commands to copy these files, and 'PROFILE.SUB' should be changed to contain the line 'APED <STARTUP'.

b). Creating a compiler system disc

The second disc will contain all the compiler programs and associated files. This is created simply by copying side 2 of the supplied master disc.

To copy this disc the supplied program 'DCOPY' must be used (on the PCW computer, the 'DISCKIT' program will not work). To use this simply type 'DCOPY' from the editor or CP/M and follow the instructions.

c). Configuration of the editor (APED)

Whilst the editor may now be used, it is important to carry out a procedure to configure it, as soon as possible, to suit the precise arrangement of the computer with which it is being used. It is not essential to do this now, but it should be done before any serious work is carried out with the editor. Full details of the options which require setting and also many other options which may be set, are given in the chapter covering the utility programs in the editor part of the manual.

THE COMPILER

1. INTRODUCTION

The Arnor C compiler consists of a number of fully integrated programs which include a powerful text editor, 'APED', a 'run time system program', 'RUNC' and a number of programs and files which comprise the compiler, linker and other ancillary programs.
This part of the manual describes the use of the various programs concerned with compiling a C program and a working knowledge of the editor (APED - Arnor Program EDitor) is assumed. It is recommended that some time is spent reading the section of the manual describing the editor and becoming acquainted with it, before using the compiler.

Arnor C is very flexible in that it is possible to use the various programs in a number of different ways and combinations.

This section starts with a description of the general principles and procedures involved in writing a C program and then progresses to detailed descriptions of the various parts of the Arnor C program and finally a description of the different ways in which the programs may be used.

a). The programs and files involved with compiling a C program

The following is a list of the programs concerned with compiling and running a C program, some or all of which will be required on all occasions.

RUNC.COM	The C run time system program.
AC.COM	A program to compile, link and run simple programs.
COMPILE.O	The compiler program.
LINK.O	The linker program.
JOIN.O	A program to join together two or more link files into one larger link file.

Library files

STDLIB.L	The full library
SMLIB.L	Small library with most of the standard functions
MINLIB.L	Minimum library containing only low level functions
MATHS.L	Mathematical function library

Header files

STDIO.H	The standard header file containing macros and structure definitions
STDLIB.H	Header file declaring functions in STDLIB.L
SMLIB.H	Header file declaring functions in SMLIB.L
MINLIB.H	Header file declaring functions in MINLIB.L
MATHS.H	Header file declaring functions in MATHS.L, and some constants

b). Writing and compiling a simple C program.

The process of writing a C program involves several stages:

1. Design the algorithms and data structures.
2. Plan the program.
3. Write the code for the program.
4. Enter the source code using the editor and save it on disc with an appropriate filename and the filename extension '.C'. For example, 'MYPROG.C'.
5. Compile the source code. This is the operation of translating the C program into a link file. The link file will be saved with the filename extension '.L'.
6. Link the file, or files. This takes one or more link files (produced by the compiler or assembler), and links them with the standard C library to produce an executable C program, which will be saved with a filename extension '.O'.
7. Run the program to test it.
8. Go back to the editor and correct the program, if necessary.
9. Repeat steps 5 to 8 as many times as necessary.

We shall not concern ourselves with items 1 to 4 and it is assumed that the source code has already been designed and entered with the editor.

The simplest way to carry out the above operations is directly from the editor and a program called AC.COM is provided expressly for the purpose of 'automating' the compilation of straightforward programs. This program is described in the next chapter.

2. AUTOMATED COMPILING

The program AC.COM is supplied in order to provide a method of compiling straightforward programs with the minimum of effort and can be used either from within the editor, or from CP/M command mode.

a). Compiling and running programs from the editor

The editor is used for entering and modifying C source code (See the section on the editor for full details of how to use it) and with straightforward programs it is most convenient to compile and run them from within the editor.

Note: If a program consists of a number of separately compiled parts which need to be linked together, it is preferable to use a different method and details are given in the chapter on the run time system.

In the simplest case, the source code for a program is entered using the editor and saved on disc. It then needs to be compiled, linked and run. These three operations can be performed as a single command, using the following command from the editor's command mode:-

AC	to compile the current text in memory

or:

AC <filename>	to compile a named file

Note: In order for AC to function, it is important that the following files are present on one of the drives:-

AC.COM	the program
RUNC.COM	the run time system
COMPILE.O	the compiler
LINK.O	the linker
STDLIB.L	the standard library

In addition, any header files required by the program must also be present.

The program will then be compiled. During the course of compiling the source code, the compiler displays a number of messages on the screen to indicate the state of progress. It is possible to redirect some or all of these messages to a file on disc, if required. Full details of the possible redirections are given in the following chapter and an Appendix.

If an error occurs during compilation, control will be returned to the editor with any text still in memory, so the error can be rapidly corrected and the process repeated.

If compilation is successful, the program will automatically be linked with the library, but if an error occurs, control will again be returned to the editor.

If both compilation and linking are successful, the program will automatically run and, on completion of the program, control will return to the editor, as before.

With the AC command, the editor provides a convenient operating environment for editing, compiling and testing programs. One of the advantages is that it is possible to use any of the editor's commands to catalogue, copy, rename or delete files, as and when required.

In the process of compilation, AC creates a number of files. Some of these files are temporary and will automatically be deleted when no longer required, but several will be retained. These will all have the same name as the original source file, but with different filename extensions:-

filename.C	is the original source file
filename.L	is the intermediate link file
filename.O	is the executable object file

b). Using AC directly from CP/M

AC may be used from CP/M in exactly the same way as from the editor, but if compilation or linking fails, or when the execution of the program is completed, control will return to CP/M command mode, rather than the editor. It is obviously more convenient to use AC from within the editor in normal circumstances.

c). Running compiled programs

Once a program has been compiled using AC and the object file has been produced, there is no need to recompile the program every time it is to be used and programs should subsequently be run by using the following command:-

RUNC <filename> (<optional parameters>)

<filename> is the name of an object file and there is no need to specify the '.O' extension. <optional parameters> is a list of one or more parameters that the program may require.

The following chapters give details of how to use the run time system to compile more complex programs.

3. THE RUN TIME SYSTEM

The run time system is the heart of the Arnor C compiler and is the program from which all compilation of code, linking of files and running of compiled programs is carried out.

Using the run time system

The run time system program is called RUNC.COM and may be used in one of two ways:-

(i) From the editor (APED)
(ii) From CP/M command mode

In addition, RUNC may be used in two different forms, interactively and passively.

Note In order to use RUNC to compile, link or join files, it is necessary for the following files to be present:-

RUNC.COM	the run time system
COMPILE.O	the compiler
JOIN.O	the joiner
LINK.O	the linker

In addition, any library files which will be required by the compiler should also be available.

a). Using the run time system interactively

When the program is used in interactive mode, it provides an environment in which programs may be compiled, linked, joined or run. When each process is completed, a return will be made to the command prompt, ready for further commands.

The program is called either from the editor, or CP/M, by typing:-

RUNC

from command mode. The screen will clear and a title message including the version number will appear at the top of the screen. A command prompt will then be displayed as in the editor. When the program is used in this way, it is in interactive mode and a variety of commands may be used.

The following 'built in' commands are available:-

A	-	to select drive A
B	-	to select drive B
M	-	to select drive M
Q	-	to quit the run time system and return to either CP/M or the editor, depending on where it was called from.

From this point, it is also possible to run any previously compiled C program (files with a '.O' filename extension).

When a command is entered, it is checked against the built in commands and if no command of that name is found, the program will search for a program of that name (with a .O extension) and execute it. It is not necessary to specify the '.O' extension, as this is assumed. Any parameters or options required by the command or program may also be entered, following the command or program name.

The program will be found whichever drive it is present on. The order of searching is M, C, A, B, D. To force a program to be taken from a particular drive, prefix the name with the drive letter, e.g. B:DUMP.

For example, entering:-

MYPROG

will run the program called 'MYPROG.O'.

COMPILE filename -L

will run the compiler and pass 'filename -L' to the compiler as parameters.

b). Using the run time system passively

So far in this chapter, RUNC has been used as an environment for issuing commands and running programs, but it is also possible to use the run time system passively.

When the command, 'RUNC', is incorporated as part of a command line from the editor or CP/M, it will operate passively and 'invisibly'. An example of this has already been encountered in the previous chapter, describing automated compilation, when 'RUNC filename' was described as the command to run a compiled program.

When the run time system is used in this way, the title is suppressed and there is no visible indication of its presence. There is no command prompt and the program specified as part of the command line will execute immediately.

The AC.COM program described earlier is directly equivalent to typing:-

RUNC COMPILE <filename> -L -R

from the editor or CP/M. (The following chapters explain the various commands to compile and link programs, as well as the '-L -R' parameters specified in the above command.)

c). Supplied programs

Arnor C is supplied with a number of programs for use with the run time system. These files all end with the extension '.O' and include:-

COMPILE	<fn> (opt)	The source code compiler.
LINK	(of=) <lfl>	The linker.
JOIN	<lf=> <lfl>	A program to join two or more link files into one link file.
* DIR	(d) or <afn>	Utility to list a disc directory.
* DUMP	<fn>	Utility to dump the contents of a file to the screen in hexadecimal and ASCII form.
* ERA	<afn>	Utility to erase files on disc.
* REN	<ofn> <nfn>	Utility to rename files.
* TYPE	<fn>	Utility to type the contents of a file on the screen in ASCII form. (Note: this is intended for use with text files and using it with program files may give unpredictable results)

Description of abbreviations

fn	a filename
afn	ambiguous filename (including wildcards)
ofn	old filename
nfn	new filename
of=	optional object filename for resulting file
lf=	link filename for resulting file
lfl	list of link files to be used
d	drive letter

Parameters in angle brackets '<..>' are mandatory, whilst those in normal brackets '(..)' are optional.

Note: the files marked with an asterisk (*) are provided for convenience and are similar to the equivalent commands in the editor with the following exceptions:-

DUMP An extra option, allows the file to be dumped from a given offset address within the file. The address is given as a C constant.

```
dump test.l 0x1000
```

ERA An extra feature is the 'verify' option. If a 'V' is typed after the filename, then for each file being erased a prompt will be given asking for confirmation.

TYPE has the same enhancement as DUMP.

d). Redirection from the command line

When a program is run, three files are automatically opened. These are called 'stdin', 'stdout' and 'stderr'. By default, 'stdin' receives keyboard input and directs it to the screen. 'stdout' directs program output to the screen and 'stderr' sends error messages to the screen.

These files may be redirected to other 'devices', as listed below.

<filename	redirects stdin to read from a file
>filename	redirects stdout to a file
>>filename	redirects stdout, appending to an existing file
#filename	redirects stderr to a file
#>filename	redirects stderr, appending to an existing file

One of the most useful of these redirections is to send any error messages produced during compilation to a file on disc. The resulting file may then be loaded into the editor for viewing at the same time as corrections are made to the source code, or may be viewed by using the 'TYPE' command.

Note: These redirections may be made from the editor or from the run time system.

e). The Screen Driver

The run time system contains specially written screen driver routines to take full advantage of the Amstrad computers and in order to optimise speed of screen writing.

In addition, provision is made to use 'windows' and special functions are provided in the libraries to permit the easy use of windows.

As a result of this, the usual methods of locating the cursor, clearing the screen etc., which are somewhat messy, are no longer necessary and the special functions provided for these purposes should be used instead. Full details of all these functions are given in the chapter on Library functions.

4. THE COMPILER

The compiler converts source code, written with the editor, into a form suitable for linking with the standard library files and other link files and, in the process, carries out certain syntax checks.

a). What the compiler does.

The compiler carries out its work in three stages.

(i) The pre-processor and lexical analysis pass.
(ii) The syntax checking and code generation pass.
(iii) The post processor pass.

(i) The pre-processor and lexical analysis pass

The initial pass creates a temporary file containing a series of lexical tokens. This means that each instruction, variable, constant or other symbol is replaced by a single number.

There are several pre-processor commands, each of which has an effect on the compiled code.

The pre-processor commands are listed below and operate in the standard manner, as described by Kernighan & Ritchie, with two small exceptions which are noted.

#assert
#define
#elif
#else
#endif
#if
#ifdef
#ifndef
#include
#line
#undef

#assert is not mentioned by Kernighan & Ritchie. Its purpose is to check the value of a constant expression and stop compilation if the value is zero.

e.g. #assert test > 0

#include differs from Kernighan & Ritchie in one respect. This concerns the characters used to enclose the filename.

#include "filename" - Searches for the file only on the current drive.
#include <filename> - Searches for the file on any drive.

Errors that may occur during the first pass include the illegal use of macros, by invoking a macro with the wrong number of parameters, for example, or not finding a specified *#include* file. An error on the first pass causes compilation to be abandoned.

(ii) The syntax checking and code generation pass

The second pass checks the program syntax and generates the compiled code. Error and warning messages are listed as the errors are found and compilation continues until the specified maximum number of errors have occurred. If an error occurs, the compiler attempts to recover by scanning forward until it finds a recognisable statement delimiter or structure. This may mean that a section of code has been skipped and subsequent error messages may occur solely as a consequence of the original error.

The names of the functions are listed as they are compiled.

(iii) The post processor pass

This pass simply writes the link file to the disc, using the name of the source file, but with an 'L' filename extension.

b). Using the compiler.

Using the compiler is straightforward and a number of options may be specified, if required, to modify the compilation process. The syntax used is as follows:-

COMPILE <source> (options)

<source> is the name of the source code file which should have been saved to disc with the filename extension 'C'.

When specifying the filename in the command, the 'C' extension may be omitted as it will be assumed by the compiler. If the source filename is omitted altogether, the compiler will prompt for a filename.

(options) may, but need not, be specified. These are entered in the form of letters, each preceded by a hyphen (-) and if more than one is specified, each separated by a space. The options available are listed below.

c). Compiler options

-d defines a macro. The text of the macro definition should follow the '-d'. Example: COMPILE PROG -dDEBUG.

Note: Macro definitions may not contain declarations in this situation. In the above example, DEBUG is defined with blank replacement text. It is not possible to use 'DEBUG 1'. This option is useful to permit conditional assembly of code, by making use of:

```
#ifdef DEBUG
{
    /* Source code to be compiled only if DEBUG is defined
    */
}
#endif
```

If '-dDEBUG' is used in the command line the conditional section of code will be included in the compilation.

-g suppresses creation of the global table. This can be used when compiling files consisting purely of data, with the advantage that the resulting object code will be smaller.

-l causes the linker to be run automatically after a successful compilation.

Note: If '-l' is used, it must be the the last of the compiler options, although it may be followed by further options relating to the use of the linker (See chapter on the linker for details).

-m sets the maximum number of errors reported by the compiler before compilation is abandoned. The default value is 20.
Example: COMPILE PROG -m1.

-q suppresses the compiler's sign on message and summary information. Error messages are still displayed.

-t specifies the drive to be used for the temporary files. By default this will be drive M on a PCW, and the current drive on a CPC6128.
Example: COMPILE PROG -tA.

-w suppresses compiler warning messages. Error messages are still displayed.

5. THE LINKER

The linker creates a single executable program from one or more link files that have been produced by the compiler.

a). What does the linker do?

The action of the linker is to resolve all external references (that is, calls to functions defined in separate source files) and to relocate all the sections of code.

For example, when a program that uses PRINTF is compiled, the compiler does not know where PRINTF is defined. It is, in fact, defined within the standard library (STDLIB.L) which is a previously compiled link file supplied with Arnor.C. The linker fills in the references to PRINTF in the main program.

It creates an executable object code file, with the same name as the first of the specified link files (unless an alternative is specified - see below), but with an 'O' filename extension.

Programs produced by the linker may then be run from within the run time system simply by typing the name of the file (without the '.O'). Alternatively they may be run from the editor or CP/M by typing RUNC followed by the program name.

b) Using the linker.

Operation of the linker is straightforward and a number of options may be specified. The syntax for the command is as follows:-

LINK (object =) <list of link files> (options)

'<object> =' is optional. If an equals sign (=) is typed after the first name, the object file produced by the linker will be saved with that name, rather than the name of the first link file. The object filename will not be treated as a link file name.

In the absence of an object filename, the first link filename will be used as the object filename, but will also be used as the name of a link file.

The < list of link files > may contain the names of one or more files with an .L filename extension. The '.L' extension may be omitted from the list of names, as it will be assumed by the linker.

All files in the link file list must be link files produced either by the compiler or by the Maxam II assembler (See later chapter for details of linking machine code files).

By default, the standard library file, STDLIB.L, is automatically linked without the need to specify it in the list of link files, unless specifically excluded with the link option '-l' (See below).

There are a number of options which modify the action of the linker and these are listed below:-

c). Linker options

-l stops the standard library (STDLIB.L) from being linked. This should be used when an alternative library (such as SMLIB.L) is used.
e.g. LINK PRIMES SMLIB -L

-n list functions whilst linking. The information is given in the following form:

< function name > < type > < code offset > < file in which defined >

< type > is a single letter, as follows:
A = an absolute value
C = a C function
D = data

-q suppresses the linker's sign on message and summary information. Error messages are still displayed.

-r automatically run the program after linking, if linking was successful. This must be the last of the linker options, although it may be followed by parameters required by the program at the time it is run.

6. LINKING MACHINE CODE PROGRAMS

This chapter describes how programs written partly in C and partly in Z80 assembly language, using the Maxam II Assembler, may be linked together.

The most common reason for wanting to write parts of a program in machine code is in order to produce functions that run faster than if they were written in C. Functions can be written to "look like" a C function and may be called from a C program in the same way as if the function had been written in C. It is also possible to call C functions from machine code.

Considerations

When writing a program in assembler that is to be linked with a C program, or vice versa, there are several things to be considered.

a). The header file
b). Function headers
c). Accessing passed parameters
d). Returning a value
e). Calling a C function

a). The header file

A header file is provided and should be included at the start of any file containing machine code functions for use with C. This file contains various macro and variable definitions.

To include this file, the following instruction should be used:

```
read "mstdio.h"
```

mstdio.h also defines an external variable *x_main* as a word. This must be present at the start of every assembled file, as it is required by the linker. It is therefore essential that the *read* directive is included before any instructions or other directives.

b). Function headers

Each function written in machine code must be preceded by a special sequence of bytes which identifies it as a function to be called by C. A macro is provided for this purpose, called *fnhdr*. Thus, a function should start as follows:

```
name
fnhdr
.. code for function ...
```

c). Accessing passed parameters

Parameters are passed to a function on the stack and must be accessed by indexing into the stack. The parameters are found on the stack in the order they are given in the function call, the first parameter starting 6 bytes above the stack pointer on entry to the routine. An extended function header macro, *xfnhdr*, will set up the HL register to point to the first parameter.

xfnhdr is defined as follows:

```
macro xfnhdr
fnhdr
ld hl,6:add hl,sp
mend
```

It is often convenient to use *xfnhdr* instead of *fnhdr*. Alternatively, it is sometimes better to use the IX or IY register to index the stack:

```
fnhdr
ld iy,6:add iy,sp
```

The function must assume that the parameters passed are of known types. The various types are stored on the stack as follows:

char	stored as an int with the high byte zero
int	2 bytes, low byte at the lower address
long	4 bytes, low byte at the lowest address
float/double	5 bytes, as follows: 4 byte signed mantissa, high byte first exponent, stored offset by 128
pointer	all pointer types are 2 byte integers

Example of accessing parameters from stack:

This example is the code for the function *movmem*, which is in the standard library. The comments at the start detail the C declarations.

```
public movmem  ;; allow name to be used elsewhere
movmem         ;; void movmem(source,dest,len)
               ;; char *source,*dest ;
               ;; unsigned len ;
               xfnhdr
               ld e,(hl):inc hl              ; low byte of source
               ld d,(hl):inc hl              ; high byte of source
               push de
               ld e,(hl):inc hl              ; low byte of dest
               ld d,(hl):inc hl              ; high byte of dest
               ld c,(hl):inc hl              ; low byte of len
               ld b,(hl):pop hl              ; high byte of len
               ex de,hl
               push hl:or a:sbc hl,de        ; check for overlapping blocks
               or a:sbc hl,bc:pop hl:jr c,mback
               ex de,hl:ldir:ret
mback          dec bc:add hl,bc:ex de,hl
               add hl,bc:inc bc:lddr:ret
```

d). Returning the function value

Any value to be returned must be stored in a memory location known as the accumulator. The location depends on the type of value being returned, and it is accessed by means of a variable and a macro defined in *mstdio.h*.

To return a value of any integer type (including pointers), it should be stored at the address *iacc*. This is a 4 byte location, though the high 2 bytes are only used when returning a long value. The value to be returned should simply be stored at *iacc*, with the low byte first.

To return a floating point value, it should be pushed onto the stack in the following order:

exponent,	stored offset by 128
mantissa,	byte 0 (most significant byte)
mantissa,	byte 1
mantissa,	byte 2
mantissa,	byte 3 (least significant byte)

The macro *retfloat* should then be used, which will pull the number off the stack into the accumulator and return from the function.

e). Calling a C function

To call a C function from machine code, the macro *cfunc* is used. This is defined in the file *mstdio.h*. *cfunc* takes two parameters, the name of the function and the number of bytes that have been pushed onto the stack as parameters.

After returning from the C function, these bytes will already have been removed from the stack.

The following example is the code for *fclose* taken from the library. In this example the function being called, *close*, is actually written in machine code, but can be called in the usual way because it starts with the function header.

```
            public fclose
            extern close

fclose      ;; int fclose(stream)
            ;; FILE *stream;

            xfnhdr
            ld e,(hl):inc hl          ; stream low byte
            ld d,(hl)                 ; stream high byte
            ld a,(de)                 ; stream -> handle
            ld hl,-1:ld (iaccl),hl
            cp &7f:ret nc             ; return -1 if not a file
            push de                   ; save address of file handle
            ld l,a:ld h,0
            push hl                   ; pass file handle on stack
                                      ; and call the close function
            cfunc close 2             ; 2 bytes on stack
                                      ; 2 bytes already removed by cfunc
            pop hl                    ; recover address of file handle
            ld a,&ff:ld (hl),a        ; stream -> handle = 0xFF
            ret
```

The corresponding C definition would be:

```
int fclose(stream)
FILE *stream:
{
    int a;
    if (stream -> handle >= 0x7F) return -1;
    a = close(fileno(stream));
    stream -> handle = 0xFF;
    return a;
}
```

7. THE JOINER

a). What the joiner does

The joiner takes two or more link files and merges them to produce a single link file. The principle use for this command is the creation of special library files and is the method that was used to produce STDLIB.L from various separately compiled and assembled routines.

b). Using the joiner

The syntax for the joiner is similar to the syntax used with the linker.

JOIN <linkfile> = <list of link files> (options)

The <list of link files> may contain the names of one or more files with an .L filename extension. The '.L' extension may be omitted from the list of names, as it will be assumed by the joiner.

All files in the link file list must be link files produced either by the compiler or by the Maxam II assembler.

There are a number of options which modify the action of the joiner which are listed below.

c). Joiner options

-n list functions whilst joining. The information is given in the following form:

<function name> <type> <code offset> <file in which defined>

<type> is a single letter, as follows:
A = an absolute value
C = a C function
D = data

-q suppresses the joiner's sign on message and summary information. Error messages are still displayed.

8. SUMMARY OF WAYS TO COMPILE

Full details of the commands and options available are given in the chapters covering each command and the object of this chapter is to show how these programs, commands and options may be used in conjunction with each other.

Arnor C consists of two main programs. The editor, 'APED.COM', is used for the creation of all source code and the run time system 'RUNC.COM', is the program through which all compilation, linking and joining of programs is carried out. A third program, 'AC.COM', is provided and this automates the processes of compiling and running simple C programs.

Once the code has been entered and saved with the editor, two options are available:-

a). Compile and run the program from within the editor.
b). Compile and run from the run time system.

a). Compile and run the program from within the editor

Programs may be compiled and run from within the editor in a number of different ways:-

(i) Using AC.COM

The simplest way to compile and run a program consisting of only a single source file is to use the AC command. The process is completely automatic and it is not possible to specify any special options. Typing:-

AC	will compile and run the file in memory
AC filename	will compile and run the specified file

When compilation is complete, the resultant link file will be automatically linked with the standard library file and, if both compilation and linking are successful, the program will automatically run on completion of linking.

When execution is complete, or if compilation, or linking fail, a return will be made to the editor and further editing of the source code may be carried out. If the source file was in memory when AC was called, it will still be present and available for immediate editing.

(ii) Using RUNC

The run time system 'RUNC.COM', may be used from within the editor and when used in this way, its presence will not be visually apparent. Alternatively, RUNC command mode can be entered from the editor and this is more suitable for complex or multiple compilations and is described in b) below.

The choice of which method to use is quite simple. If used passively, from the editor's command mode, as soon as that command sequence has been completed the editor will be re-entered. If RUNC is being used instead of the AC command to enable special options to be selected when compiling a single file, this will probably be the most convenient option. The syntax and options vary according to which command is used and the simplest way is to give some examples:-

RUNC COMPILE filename -L -R

will compile the file called filename, the '-L' option will cause it to be linked and the '-R' will be passed to the linker and is the option to run the resultant program. On completion of running the program, a return will be made to the editor. This is effectively what happens when the AC command is used.

RUNC LINK obfile = smlib source1 source2 -L

will link the already compiled link files 'source1' and 'source2' with the small library file. The '-L' option will suppress the linking of the standard library file, which is normally carried out automatically. The resultant object file will be saved with the 'obfile' filename.

If a number of compilation commands are required, it is more convenient to enter the run time system and remain there until all work has been completed and this is covered in the next section.

b). Compile and run from the run time system

The run time system may be entered from the editor by typing:-

RUNC

The run time system program will load and the status message will appear at the top of the screen, with a command mode prompt and cursor beneath. From this point, commands may be entered in exactly the way that they were described in the previous section, with the exception that the 'RUNC' should be omitted. All options for the compiler, linker and joiner are available and the various modules may be automatically linked, as before.

Care should be taken with the order in which the options are specified. The '-L' option, to call the linker, must be the last of the compiler options and '-R', to run the object file, must be the last of the linker options. It is quite permissible to use other options after the compiler '-L', or linker '-R', as long as the options which follow relate to the subsequent command.

COMPILE filename -Q -L -Q -R datafile

will compile the file called filename, suppressing the opening message and summary and automatically link the resultant file with messages suppressed and finally run the object file, passing 'datafile' as a parameter required by the object file.

JOIN linkfile = link1 link2 -Q

will join the link files, 'link1' and 'link2' and save them as one link file called 'linkfile' and the opening message and summary will be suppressed.

c). Using RUNC from CP/M command mode

The run time system can be used from CP/M in exactly the same ways as described for the editor. RUNC may be used on its own to enter the run time system, or with the additional commands and options to compile or link a program. The only difference in operation is that when the command is completed or terminated, a return will be made to CP/M.

d). Running compiled programs

Previously compiled programs may be run without the need for recompilation, at any time, either from CP/M or the editor by using the following command syntax:-

RUNC progname (parameters)

(parameters) is optional and they are only necessary when a program requires parameters passing in the command tail. Alternatively the program may be run from within the run time system by omitting 'RUNC'.

A9. LIBRARY FUNCTIONS

This chapter gives full details of each library function. Functions may be described as being:-

Standard	- as mentioned by Kernighan and Ritchie, or found in the majority of C systems.
Common	- found in many C systems.
Arnor C	- functions specially written for Arnor C. Most of the functions concerned with screen handling and windows are of this sort.

Note: If programs are intended to be made 'portable' to other machines, the Standard functions should be used wherever possible.

Each function lists the parameters and the declarations required. The description covers whether the function is Standard or not, which library file it is in and describes its purpose. The results returned, if any, are also described.

All functions except the mathematical functions are contained in the library, *STDLIB.L* which is used by default. The mathematical functions are marked 'Maths' in the list below. The small library, *SMLIB.L* contains a subset of the functions in *STDLIB.L*, and these functions are marked 'Small'.

Additionally some functions are implemented as macro definitions in the header file, *STDIO.H*, and these are marked 'Macro'.

A summary of all functions can be found as Appendix 3.

Example programs

Programs showing examples of the use of many of the library functions are provided on the disc. Consult the file 'README' for details.

abort
void abort()

Description: Common

Terminates the program without closing files. It is the same as *_exit(1)*.

Returns: doesn't return.

abs
abs(x)

Description: Standard, Macro

Takes the argument *x* and returns the absolute value of it. Note that this function is declared as a macro, and will work with any numerical argument.

Returns: the absolute value of *x*.

acos
double acos(x)
double x;

Description: Common, Maths

Returns the inverse cosine of *x*, in radians. *x* must be in the range -1.0 to +1.0.

Returns: the inverse cosine of the argument, *x*, in radians.

asin
double asin(x)
double x;

Description: Common, Maths

Returns the inverse sine of the argument *x*, in radians. *x* must be in the range -1.0 to +1.0.

Returns: the inverse sine of the argument *x*, in radians.

atan
double atan(x)
double x;

Description: Common, Maths

Returns the inverse tangent of the argument, *x*, in radians. *x* may be any real number.

Returns: the inverse tangent of the argument, *x*, in radians.

atan2
double atan2(x,y)
double x,y;

Description: Common, Maths

Returns the inverse tangent of *y/x* in radians. *y* and *x* may be any real numbers.

Returns: the inverse tangent of *y/x* in radians.

atof
double atof(s)
*char *s;*

Description: Standard

Converts the string pointed to by *s* into a double. The string may have leading spaces and tabs, and follows the C syntax for a floating point constant. Conversion stops at the first inappropriate character.

Returns: floating point value of the string, or 0 if no number recognised.

atoi
int atoi(s)
*char *s;*

Description: Standard

Converts the string pointed to by *s* into an integer. The string may have leading spaces and tabs, and follows the C syntax for an integer constant. Conversion stops at the first inappropriate character.

Returns: integer value of the string, or 0 if no number recognised.

atol
long atol(s)
*char *s ;*

Description: Standard

Converts the string pointed to by *s* into a long integer. The string may have leading spaces and tabs, and follows the C syntax for an integer constant. Conversion stops at the first inappropriate character.

Returns: long integer value of string, or 0 if no number recognised.

bdos
long bdos(c,de)
int c, de ;

Description: Arnor C, Small

Calls the BDOS routine specified by *c*, passing *de* in the *DE* register. Returns a long composed of the values returned from the BDOS routine; the low integer holding the value returned in A, the high integer the value returned in HL.

Note: the Arnor C screen output functions do not use the BDOS routines, and so the BDOS console output routines should not be used. Similarly, the BDOS console input routines are redundant (although their use should cause no problems).

Returns: a long composed of the values returned from the BDOS routine.

bios
int bios(n)
int n ;

Description: Arnor C

Calls the BIOS routine *n*, and returns the value returned in the A register.

Returns: the value passed back by the BIOS routine.

busypr

int busypr()

Description: Arnor C

Tests whether the printer is ready.

Returns: *TRUE* if ready, or *FALSE* if not.

call

void call(addr,regs)
unsigned addr;
int regs[6];

Description: Arnor C

This is a general purpose routine for calling machine code at a specified address, and is provided only for specialised use. *regs* is a vector, in which is passed the values to be passed to the routine in the Z80 registers. The registers are stored as follows (each with the low byte first):

regs[0]	AF
regs[1]	BC
regs[2]	DE
regs[3]	HL
regs[4]	IX
regs[5]	IY

Returns: *regs* will be updated to contain the values of the registers on exit from the machine code routine.

calloc

*char *calloc(number,size)*
unsigned number,size;

Description: Standard

Allocates a block of memory of *number * size* bytes. The memory is initialised to zeros.

Returns: a pointer to the start of the block, or *NULL* (0) if unsuccessful.

ceil
double ceil(d)
double d ;

Description: Common, Maths

Returns the smallest integer greater than or equal to the argument.

Returns: the smallest integer greater than or equal to the argument.

clearerr
void clearerr(fp)
*FILE *fp ;*

Description: Standard, Macro

Clears the error indication for the stream *fp*.

Returns: nothing.

close
int close(handle)
int handle ;

Description: Standard, Small

Closes the file with specified *handle*.

Returns: 0 if successful, *ERROR* (-1) if an error occurred.

cos
double cos(x)
double x ;

Description: Common, Maths

Returns the cosine of the argument, which is in radians.

Returns: the cosine of the argument, in radians.

cosh
double cosh(x)
double x;

Description: Common, Maths

Returns the hyperbolic cosine of the argument.

Returns: the hyperbolic cosine of the argument.

creat
int creat(name,pmode)
*char *name;*
int pmode;

Description: Standard, Macro

Creates a file with the specified name. This function is included for compatibility with early versions of C. *pmode* is ignored. New programs should use either *fopen* or *open* to create new files. If successful, the file is opened for writing, and a file handle is returned.

Returns: a file handle if successful, *ERROR* (-1) if not.

cursoff
void cursoff()

Description: Arnor C, Small

Turns off the cursor display.

Returns: nothing.

curson
void curson()

Description: Arnor C, Small

Turns on the cursor display. This routine is not usually needed because the cursor is automatically turned on by *getch* when waiting for a key, and turned off afterwards.

Returns: nothing.

drsearch
int drsearch(fname)
*char *fname ;*

Description: Arnor C, Small

Searches for the specified file on all drives, in the order M,C,A,B,D. If the file is found on a drive, that drive is selected, otherwise the selected drive remains the same.

Returns: *ERROR* (-1) if an error occurs, otherwise 0.

ecvt
*char *ecvt(val,ndig,dp,sign)*
double val ;
*int ndig,*dp,*sign ;*

Description: Common

Converts the double *val* into its ASCII representation (printf() %e format) with *ndig* significant digits. The position of the decimal point relative to the beginning of the string is stored in **dp*. If **dp* is negative, the decimal point is positioned that many places to the left of the string. **sign* will contain zero if *val* is non-negative, or a non-zero value if *val* is negative. The string is written into a static workspace, which is used by every call to *ecvt* or *fcvt*.

Returns: a pointer to the string of digits.

execv
void execv(name,argv)
*char *name, *argv[] ;*

Description: Common

Constructs a string from the strings pointed to by *name*, *argv[0]*, *argv[1]*,...(separating them by spaces), and passes the result to *_exec*. If there are n arguments, *argv[n]* must be *NULL* (0). The maximum possible length for the resulting string is 255 characters.

Returns: nothing.

exit
void exit(ecode)
int ecode ;

Description: Standard, Small

Flushes all output buffers, closes all output files and returns from the program with an exit status given by *ecode*. By convention *ecode* is zero to indicate a normal end of program and non zero to indicate an error. The error code is used to set the CP/M return code and may be accessed by other programs.

Returns: doesn't return.

exp
double exp(x)
double x ;

Description: Common, Maths

Returns the natural exponent of the argument *x* (e to the power of *x*).

Returns: the natural exponent of the argument *x*.

fabs
double fabs(d)
double d ;

Description: Common, Maths

Returns the absolute value of the floating point argument, *d*.

Returns: the absolute value of the argument.

fbinary
int fbinary(fp)
*FILE *fp ;*

Description: Arnor C

The translation mode for the stream *fp* is set to binary.

Returns: 0 is returned if successful, *ERROR* (-1) if the stream *fp* is not in use.

fclose
int fclose(fp)
*FILE *fp ;*

Description: Standard, Small

Closes the file whose file descriptor is pointed to by *fp*. Any output buffer for *fp* is flushed before closing.

After redirecting one of the standard streams (*stdin*, *stdout*, *stderr*, *stdprn*), *fclose* may be used to restore the default setting, e.g. *fclose(stdin)* will cause input to be taken from the keyboard again.

Returns: 0 if the stream is successfully closed, *ERROR* (-1) on error.

fcvt
*char *fcvt(val,ndig,dp,sign)*
double val ;
*int ndig,*dp,*sign ;*

Description: Common

The same as *ecvt* except that *ndig* is the number of digits after the decimal point (printf() %f format).

Returns: a pointer to the string.

feof
int feof(fp)
*FILE *fp ;*

Description: Standard, Small

Tests whether the end of file has been reached on stream *fp*.

Returns: 0 if end of file has not been reached, non-zero if currently at the end of the file, or *ERROR* (-1) if an error occurs.

ferror
int ferror(fp)
*FILE *fp ;*

Description: Standard, Macro

Returns the number of the last error that occurred whilst accessing the stream *fp*. If no errors have occurred zero is returned. Any disc I/O error that occurs causes the error flag to be set, and this flag may be checked at any later time, using *ferror*.

The error codes returned by *ferror* can be used to determine the function in which the error occurred. Note that these values are specific to Arnor C. Programs which need to be portable may only assume that a non zero code signifies an error of some kind. The error codes are as follows:

1	fclose
2	feof
3	fflush
4	fgetc
5	fputc
6	fread
7	fwrite
8	fseek
9	ftell
10	ungetc

Other functions call one of the above, for example *fgets* calls *fgetc*.

Returns: 0, if no errors, otherwise the number of last error.

fflush
int fflush(fp)
*FILE *fp ;*

Description: Standard

Flushes all file buffers. If a file is open for writing, the buffer is written. If it is open for reading, the buffer is cleared. The parameter *fp* is included for compatibility with other versions of C, but is not used.

Returns: 0, if buffers were successfully flushed, otherwise *ERROR* (-1).

fgetc
int fgetc(fp)
*FILE *fp ;*

Description: Standard, Small

Reads the next character from the stream *fp*.

Returns: the character read, or *EOF* (-1) if an error occurred or end of file was encountered.

fgets
*char *fgets(s,n,fp)*
*char *s ;*
int n ;
*FILE *fp ;*

Description: Standard, Small

Reads characters from the stream *fp* into the string pointed to by *s*. Reading stops when *n-1* characters or a newline are read, end of file is encountered or an error occurs. If a newline is read, it is included in the string. The string read is terminated with a zero.

Returns: *s* if successful, *NULL* (0) if an error occurred or end of file was encountered.

fileno
int fileno(fp)
*FILE *fp ;*

Description: Standard, Macro

Returns the file handle for the given stream. File handles are between 0 and *MAXFILES-1*. Other values returned have the following meanings:

0xFF	stream not in use
0x80	stream takes input from keyboard and outputs to screen
0x7F	stream outputs to parallel printer, no input

Returns: the file handle for the given stream. See also above.

filesize

long filesize(name)
*char *name ;*

Description: Common

Returns the length of the file in bytes.

Note: under CP/M exact file sizes are not stored, and so the result will be a multiple of 128 bytes.

Returns: file size in bytes (multiples of 128 bytes), *ERROR* (-1) if unsuccessful.

firmware

void firmware(addr,regs)
unsigned addr ;
int regs[6] ;

Description: Arnor C

Calls a routine in the extended firmware jump block, or the firmware ROM (CPC6128 only), using the same format as *call* for the parameters. For details of these routines, consult the following books:

"The Digital Research CP/M Plus Manual for Amstrad PCW8256/CPC6128"
"CPC464/664/6128 Firmware" (Soft 968)

Returns: nothing.

floor

double floor(d)
double d ;

Description: Common, Maths

Returns the greatest integer less than or equal to the argument.

Returns: the greatest integer less than or equal to the argument.

fmod

double fmod(x,y)
double x, y ;

Description: Common, Maths

Returns the floating point remainder. If y is zero, x is returned. Otherwise the returned value z has the same sign as x, is less than y, and satisfies $x = i * y + z$, where i is an integer.

Returns: the floating point remainder.

fopen

*FILE *fopen(name,mode)*
*char *name, *mode ;*

Description: Standard, Small

Opens the file *name*. *mode* is a zero-terminated character string indicating how the file is to be opened. Possible values are :

"r" Open for reading, position at start of the file. The file must exist.
"w" Open for writing, if the file exists it is truncated (the contents are discarded), otherwise the file is created.
"a" Open for appending, if the file exists it is opened for writing at the end of the file, otherwise the file is created.
"r+" Open for reading and writing, position at start of the file. The file must exist.
"w+" Open for reading and writing, if the file exists it is truncated, otherwise the file is created.
"a+" Open for reading and appending, if the file exists position at end of file, otherwise create the file.

All open modes allow random access of files with *fseek*. In this implementation all modes allow reading of the file, thus "w" and "w+" are equivalent, as are "a" and "a+". *fopen* returns a pointer to the file if successfully opened, a *NULL* pointer (0) in case of error.

Note that modes "a" and "a+" cause a seek to the end of file on opening the file, and not before every write operation. This is because the exact end of file is not maintained by CP/M. However if a file was written sequentially using Arnor C, opening it in append mode will correctly position the file pointer after the last byte that was written to the file.

Note: binary and text files

In text (or translated) mode carriage returns are ignored on input, and on output a line feed causes the two characters carriage return, line feed to be written. The CTRL-Z character (26) is taken to mean end of file when using text mode. The default mode is text mode.

This translation is performed by the standard character I/O routines, *fputc* and *fgetc*. These are called by *fgets*, *fputs*, *fprintf* and *fscanf*. Data input or output using *fread* and *fwrite* is not translated because these routines use fast block I/O.

In binary (or untranslated) mode the data is read from the file and passed to the program exactly as it is. To open a file in binary mode a "b" must be the last character in the *mode* string, e.g. *fopen(infile,"r+b")* or *fopen("data.bin","wb")*. The routines *fbinary* and *ftext* allow the translation mode to be changed after a file has been opened.

Returns: a pointer to the file, or *NULL* (0) pointer if unsuccessful.

fprintf
int fprintf(fp,form,args...)
*char *form ;*
*FILE *fp ;*

Description: Standard, Small

Writes a string to the stream *fp* using the format string *form* and inserting the arguments accordingly. See *printf* for details on how the format string is interpreted.

Returns: 0 if successful, otherwise *ERROR* (-1).

fputc
int fputc(c,fp)
int c ;
*FILE *fp ;*

Description: Common, Small

Writes the character *c* to the stream *fp*. Returns *c* if successful, *ERROR* (-1) on error.

Returns: the character, if successful, *ERROR* (-1) on error.

fputs

int fputs(s,fp)
*char *s ;*
*FILE *fp ;*

Description: Standard, Small

Writes the string *s* (excluding the terminating zero) to the stream *fp*.
Returns: 0 if successful, non-zero if an error occurred.

fread

int fread(buf,size,n,fp)
*char *buf ;*
unsigned size, n ;
*FILE *fp ;*

Description: Standard, Small

Reads *n* items, each *size* bytes long, from the stream *fp* and stores them in the buffer *buf*. The number of items read is returned, or *ERROR* (-1) is returned if an error occurred.

Returns: the number of items read, or *ERROR* (-1) if an error occurred.

free

void free(p)
*char *p ;*

Description: Standard, Small

The block pointed to by *p* is freed for re-use by the memory allocator. *p* must be a value that was returned by a call to *malloc* or *calloc*. If *p* is not such a value, the effect is not defined.

Returns: nothing.

freopen
*FILE *freopen(name,mode,fp)*
*char *name, *mode ;*
*FILE *fp ;*

Description: Standard

The stream *fp* is closed, and is replaced by the file *name* with the specified open mode. This routine is most commonly used for re-directing *stdin* or *stdout*.

Returns: *fp*, if successful, otherwise *NULL* (0).

frexp
double frexp(x,p)
double x ;
*int *p ;*

Description: Common, Maths

Splits *x* into mantissa and exponent. The exponent is stored at the address pointed to by *p*, and the mantissa is returned by the function.

Returns: mantissa of *x*.

fscanf
int fscanf(fp,form,args...)
*char *form ;*
*FILE *fp ;*

Description: Standard

Reads a string from the stream *fp* using the format string *form*, extracting the arguments accordingly and storing them through the *args* pointers. See *scanf* for information on how the string is interpreted.

Returns: the number of arguments assigned, or *EOF* (-1) on error or end of file.

fseek
long fseek(fp, offset, origin)
*FILE *fp ;*
long offset ;
int origin ;

Description: Standard, Small

The file pointer for the stream *fp* is moved to the position *offset* relative to the specified origin. The values of *origin* have the following meanings:

0 - the start of the file
1 - the current position
2 - the end of the file

fseek should not be used to seek beyond the end of the file.

Note: If the file is opened in text mode, care must be taken with the offset values. The offset refers to the actual number of bytes in the file, not the number of bytes that would be read in translated mode. As a general rule, only use values returned by *ftell* as offsets when using text mode. Alternatively use only binary mode to avoid this problem.

Care should be taken if seeking from the end of a file. Since CP/M stores files in 128 byte blocks, the exact end of the file will not be known. In general it is advisable to seek either from the start or the current position.

Returns: 0 if successful, otherwise *ERROR* (-1).

ftell
long ftell(fp)
*FILE *fp ;*

Description: Standard, Small

Returns the offset of the current pointer into the file associated with stream *fp*. This value is suitable for use with *fseek* with origin 0. If an error occurs *ftell* returns (long) -1. See note under *fseek* concerning files opened in text mode.

Returns: the offset of the current pointer into the file, or *ERROR* (-1), on error.

ftext
int ftext(fp)
*FILE *fp ;*

Description: Arnor C

The translation mode for the stream *fp* is set to text.

Returns: 0 if successful and *ERROR* (-1) if the stream *fp* is not in use.

fwrite
int fwrite(buf,size,n,fp)
*char *buf ;*
unsigned size, n ;
*FILE *fp ;*

Description: Standard, Small

Writes *n* items, each *size* bytes long, to the stream *fp*. The data is taken from the buffer *buf*.

Returns: the number of items written, or *ERROR* (-1) if an error occurred.

gcvt
*char *gcvt(val,ndig,buf)*
double val ;
int ndig ;
*char *buf ;*

Description: Common

This is equivalent to *sprintf(buf,"%*g",ndig,val)*.

Returns: 0, if successful, otherwise *ERROR* (-1).

getc
int getc(fp)
*FILE *fp;*

Description: Standard, Small

The same as *fgetc*.

Returns: the character read, or *EOF* (-1) if an error occurred or end of file was encountered.

getch
int getch()

Description: Common, Small

Gets a character from the keyboard, but without echoing to the screen.

Returns: the character read, or a value of *ERROR* (-1), on error.

getchar
int getchar()

Description: Standard, Macro

Gets a character from the standard input. This function is equivalent to getc(*stdin*).

Returns: the character read, otherwise *ERROR* (-1) on error.

getche
int getche()

Description: Common, Small

Gets a character from the keyboard, and echoes it to the screen.

Returns: the character read, or a value of *ERROR* (-1) on error.

getcurs
void getcurs(col,row)
*int *col,*row;*

Description: Arnor C, Small

Returns the current cursor position in the variables whose addresses are passed. The top left of the window is position (0,0).

Returns: the current cursor position in the variables whose addresses were passed.

getdrive
int getdrive()

Description: Arnor C

Returns the currently selected drive, 0 = A, 1 = B, 12 = M.

Returns: the currently selected drive, 0 = A, 1 = B, 12 = M.

gets
*char *gets(str)*
*char *str;*

Description: Standard, Small

Reads characters from *stdin* into the string *str* until a newline is read or end of file is encountered. The string is terminated with a zero. *str* must be large enough to hold the resulting string.

Returns: *str* if successful, or *NULL* (0) on error.

getw
int getw(fp)
*FILE *fp;*

Description: Standard, Small

Reads a two byte integer from the stream *fp* (low byte first), and returns the value. Since all return values are valid, *ferror* should be used to test for errors.

Returns: value read from stream *fp*.

getwin

void getwin(x1,y1,x2,y2)
*int *x1, *y1, *x2, *y2 ;*

Description: Arnor C, Small

Returns the absolute physical screen coordinates of the current window. The top left is (*x1,y1*), the bottom right is (*x2,y2*). The addresses of 4 integer variables must be passed. Top left of the screen is (0,0).

Returns: absolute screen coordinates of current screen window.

index

*char *index(s,c)*
*char *s ;*
char c ;

Description: Standard, Small

index is another name for *strchr.*

Returns: pointer to character, or if not found, *NULL* (0).

inp

char inp(addr)
unsigned addr ;

Description: Common

Reads a byte from the specified I/O port.

Returns: value read from port.

invoff

void invoff()

Description: Arnor C, Small

Turns off screen 'inverse mode'. See *invon* below.

Returns: nothing.

invon

void invon()

Description: Arnor C, Small

Selects 'inverse mode'. All characters printed subsequently will have the background and foreground reversed.

Returns: nothing.

is???? functions

	Returns *TRUE* if
int isalnum(c)	c is a letter or digit
int isalpha(c)	c is a letter
int isascii(c)	c is less than 128
int iscntrl(c)	c is a control character (127 or less than 32)
int isdigit(c)	c is a decimal digit
int isgraph(c)	c is a printable character and not space
int islower(c)	c is a lower case letter
int isprint(c)	c is a printable character (including space)
int ispunct(c)	c is a punctuation character (isprint and not isalnum)
int isspace(c)	c is a space, tab, return, line feed or form feed
int isupper(c)	c is an upper case letter
int isxdigit(c)	c is a hexadecimal digit

char c ;

Description: Standard, Macro

These functions all test a character argument. These are implemented as macros and functions. It may be necessary to use the function forms if use of the macro causes unwanted side effects. In order to use the function version *#undef* should be used to remove the appropriate macro definition.

Returns: non zero (*TRUE*) if the test succeeds, and zero (*FALSE*) if it fails.

kbhit
int kbhit()

Description: Common, Small

Tests whether a key has been pressed, returns *TRUE* if so, *FALSE* if not. If *TRUE* is returned, the character is not read but will be returned by the next call to *getch* or *getche*.

Returns: *TRUE* if key pressed, otherwise *FALSE*.

ldexp
double ldexp(x,i)
double x;
int i;

Description: Common, Maths

Calculates the value of $x *$ (2 to the power of i).

Returns: the value of $x *$ (2 to the power of i).

log
double log(x)
double x;

Description: Common, Maths

Calculates the natural logarithm of the argument, which must be positive.

Returns: the natural logarithm of the argument.

log10
double log10(x)
double x;

Description: Common, Maths

Calculates the logarithm to base 10 of the argument, which must be positive.

Returns: the logarithm to base 10 of the argument.

longjmp

void longjmp(env,val)
jmp_buf env ;
int val ;

Description: Standard, Small

Performs a 'goto' between functions. This is particularly useful for dealing with errors encountered in low level subroutines of a program. *longjmp* restores the environment saved by the last call to *setjmp* with the same *env* argument. After *longjmp* is completed, program execution continues as if the corresponding call to *setjmp* had just returned the value *val*. All accessible data have values as of the time *longjmp* was called. It is important not to call *longjmp* before a call has been made to *setjmp*, and it must also not be called after the function that most recently called *setjmp* has returned.

Returns: returns *val*, which must not be zero.

lseek

long lseek(h,offset,origin)
int h, origin ;
long offset ;

Description: Standard, Small

The pointer into the file whose handle is *h* is moved to position *offset* relative to the specified origin. The values of *origin* have the following meanings:

0 - the start of the file
1 - the current position
2 - the end of the file

Returns: the offset in bytes of the new position from the beginning of the file, or *ERROR* (-1) if an error occurred.

mallinfo

*mallblock *mallinfo()*

Description: Arnor C, Small

Returns a pointer to a structure giving information about the current state of the memory allocation. The structure is as follows:

```
struct
{
        unsigned int limalloc ;     /* size of largest free block in bytes */
        unsigned int nfree ;        /* total free space in bytes */
        unsigned int nalloc ;       /* total allocated space in bytes */
        int nblocks ;               /* the number of free blocks */
} ;
```

Returns: a pointer to a structure giving information about the current state of the memory allocation.

malloc

*char *malloc(nbytes)*
unsigned nbytes ;

Description: Standard, Small

A block of memory of size *nbytes* is allocated from the heap. There is a small memory overhead for each allocated block.

Returns: a pointer to the block of memory allocated, or *NULL* (0) if there is not enough memory.

matherr

matherr(e)
*struct exception *e ;*

Description: Common, Maths

matherr() is called by functions in the mathematical library when errors are detected. Users may use the library supplied *matherr()* function or define their own procedures for handling errors by including a function named *matherr()* in their programs. *matherr()* must be of the form described above. A pointer to the exception structure will be passed to the user supplied *matherr()* function when errors occur. The structure is defined in the header file *<arnor.h>*.

Returns: 0 if the error was handled correctly, 1 otherwise.

max

max(val1,val2)

Description: Common, Macro

Determines the higher of the two values passed. This function is implemented as a macro and can be used with any types.

Returns: the higher of the two values passed.

memchr

*char *memchr(m,c,n)*
*char *m;*
char c;
unsigned n;

Description: Common

memchr searches *n* bytes of memory starting at *m* for the first occurrence of the character *c*.

Returns: a pointer to the character, or *NULL* (0) if not found.

memcmp

int memcmp(m1,m2,n)
*char *m1,*m2;*
unsigned n;

Description: Common

Compares successive bytes of memory starting at *m1* and *m2*, until either the bytes have different values, or *n* bytes have been compared.

Returns: a positive, zero or negative value depending on the result of the comparison of the first pair of different bytes (positive if *m1* is greater).

memcpy
*char *memcpy(m1,m2,n)*
*char *m1,*m2 ;*
unsigned n ;

Description: Common

Copies *n* characters from *m2* to *m1*.

Returns: *m1*

memset
*char *memset(m,c,n)*
*char *m ;*
char c ;
unsigned n ;

Description: Common

Sets *n* bytes starting at *m* to the value *c*.

Returns: *m*

min
min(val1,val2)

Description: Common, Macro

Determines the lower of the two values passed. This function is implemented as a macro and can be used with any types.

Returns: the lower of the two values passed.

modf
double modf(x,p)
*double x,*p ;*

Description: Standard, Maths

Calculates the fractional value of *x* with the same sign as *x*. The integral part of *x* is stored at the location pointed to by *p*.

Returns: the fractional value of *x* with the same sign as *x*.

movmem

void movmem(source,dest,len)
*char *source, *dest;*
unsigned len;

Description: Common

Moves *len* bytes of memory from *source* to *dest*. The two memory blocks may be overlapping.

Returns: nothing.

open

int open(path,mode)
*char *path;*
int mode;

Description: Standard, Small

The file is opened. The mode is constructed by an 'inclusive or' of the following constants. The values of these constants are defined in the file *STDIO.H*.

Note: Not all combinations are permitted.

O_RDONLY	open for reading only
O_WRONLY	open for writing only
O_RDWR	open for reading and writing
O_CREAT	create file if it does not exist
O_TRUNC	truncate the file if it exists
O_APPEND	seek to the end of file after opening. This does not cause a seek to the end of file before every write operation, *fopen* must be used with mode "a" or "a+" to achieve this.
O_EXCL	an error occurs if both O_CREAT and O_EXCL are set.

Returns: the file pointer if successful or *ERROR* (-1) if an error occurred.

outp
void outp(addr,c)
unsigned addr;
char c;

Description: Common

Writes a byte, *c*, to the specified I/O port. The computer manual should be consulted for the port addresses.

Returns: nothing.

peek
char peek(addr)
unsigned addr;

Description: Common

Reads the value of the byte stored at memory address *addr*.

Returns: the value of the byte stored at memory address *addr*.

poke
void poke(addr,c)
unsigned addr;
char c;

Description: Common

Stores the byte *c* at the specified memory address.

Warning: this is potentially a very dangerous function and is intended for specialised use only.

Returns: nothing.

pow
double pow(x,y)
double x,y ;

Description: Standard, Maths

Calculates the value of *x* to the power of *y*.

Returns: the value of *x* to the power of *y*.

prch
int prch(c)
char c ;

Description: Arnor C

Sends a character to the printer.

Returns: *TRUE* if successful, otherwise *FALSE*.

printf
int printf(form,arg1,arg2,...)
*char *form ;*

Description: Standard, Small

printf is a general purpose formatted print routine which sends its output to stdout. Arguments are interpreted according to the zero-terminated format string. The format string is a sequence of characters with embedded conversion commands. Conversion commands are prefixed by '%'. Characters that are not part of the conversion command are output.

The general format of a conversion command is:

%fw.plc

f, w, p and l are optional.

c (conversion character) specifies the type of argument. Each conversion command causes the next argument to be read from the argument list.

The following conversion characters are provided:

d decimal signed integer
u unsigned decimal integer
o octal integer
x hexadecimal integer (using lower case letters)
X hexadecimal integer (using upper case letters)

With each of the above, the precision defaults to 1 if not specified.

c single character

s string, terminated by zero.

e the argument is a double and is displayed as an exponential floating point number. The number is printed using scientific notation, with one digit before the decimal point. The number of digits after the point is determined by the precision (see below). The precision defaults to 6 if not given. The exponent is preceded by 'e'.

E the same as e, except that the exponent is preceded by 'E'.

f the argument is a double and is displayed without exponent. The precision determines the number of digits printed after the decimal point, and this defaults to 6.

g uses e or f format, whichever is shorter.

G uses E or f format, whichever is shorter.

% the '%' character is printed.

f (flag) is one of '-', '+', '#' or space. These have the following meanings:
- left justify the output
+ forces the sign of a number to be printed
space forces positive numbers to start with a space
with o (octal) conversions a leading zero is added
with x (hexadecimal), '0x' or '0X' is added at the start
with floating point conversions (E, f, g, G) it forces a decimal point to be printed

w (width) is a decimal number which specifies the minimum field width for the conversion, that is the minimum number of characters that will be printed. If the number of characters resulting from the conversion is less than this number, the field is padded with spaces (unless the first digit of the width is a zero, in which case it is padded with zeros).
If the number of characters required is more than the width, the output is printed in the minimum space.
If '*' is specified, the width is taken from the next argument.

.p (precision). A decimal point followed by a decimal number specifies the precision of a floating point conversion, or the maximum field width for a string. For the g and G conversions, the precision is the maximum number of significant digits printed. For e, E, and f it gives the number of digits to be printed after the decimal point. If the precision starts with a 0, the output is padded with zeros.
If '*' is specified, the precision is taken from the next argument.

l (long) causes the argument to be taken as type long, if the conversion character is o, u, x, X, i or d, otherwise it is ignored.

Note: the version of *printf* supplied in the small library does not support any of the floating point options.

Returns: the number of characters written. If an output error occurred, a negative integer is returned.

putc
int putc(c,fp)
char c ;
*FILE *fp ;*

Description: Standard, Small

The same as *fputc*. Writes a character, *c*, to the stream specified by *fp*.

Returns: the character, if successful, otherwise *ERROR* (-1) on error.

putch

void putch(c)
char c ;

Description: Common, Small

Writes a character to the screen, even if *stdout* has been re-directed. Characters with an ASCII value less than 32 are not displayed, but treated as control codes. The following codes are recognised, others are not defined.

1	new line, unless the cursor is in column 1
7	sounds a beep
8	backspace
9	forward space
10	line feed
11	reverse line feed
12	clear current window
13	carriage return
18	clear to end of line
24	toggle inverse on/off
30	home the cursor

Returns: nothing.

putchar

int putchar(c)
char c ;

Description: Standard, Macro

Writes a character to the stream *stdout*.

Returns: the character just written, or *ERROR* (-1) if an error occurs.

puts

int puts(s)
*char *s ;*

Description: Standard, Small

Writes the string *s* to *stdout* (without the terminating zero), followed by a newline.

Returns: 0 if successful, otherwise non-zero. *puts()* returns *ERROR* (-1) when an error occurs.

putw
int putw(x,fp)
*FILE *fp ;*
int x ;

Description: Standard, Small

Writes a two byte integer to the file specified by the stream *fp*, using *putc*. The number is output with the low byte first.

Returns: the last character output by *putc*, or *ERROR* (-1) in case of error.

rand
int rand()

Description: Standard, Maths

Generates a pseudo random number in the range 0 to 32767.

Returns: a random number in the range 0 to 32767.

rdmatrix
void rdmatrix(c,buf)
char c ;
char buf[8] ;

Description: Arnor C, Small

Returns the matrix for the character *c* in the 8 byte buffer. The bytes returned are the bit image data for each row of the character matrix, starting with the top row. Within each byte, bit 7 is the leftmost pixel.

Returns: the matrix for the character *c* in the 8 byte buffer.

read
int read(h,buf,n)
int h, n ;
*char *buf ;*

Description: Standard, Small

Reads *n* bytes from the file with handle *h* and stores them in the buffer *buf*.

Returns: the number of bytes transferred, *ERROR* (-1) if an error occurs. 0 indicates that the end of file has been reached.

rename
int rename(oldn,newn)
*char *oldn, *newn ;*

Description: Common

Changes the name of a file from *oldn* to *newn*. Both *oldn* and *newn* may contain drive and path names, but both names must refer to the same disc.

Returns: 0 if the filename was successfully changed, otherwise *ERROR* (-1) if the rename failed.

rewind
void rewind(fp)
*FILE *fp ;*

Description: Standard

Moves the file pointer, *fp*, to the start of the file. This is equivalent to *fseek(fp,0L,0)*.

Returns: nothing.

scanf
int scanf(format,arg1,arg2...)
*char *format;*

Description: Standard

scanf is the analogue of *printf* for formatted input. Characters are read from *stdin*, interpreted according to the format specification, and stored in the arguments, which must be pointers to appropriate variables.

The format string may contain the following:

(i) white space characters (spaces, tabs and newlines). These are ignored.
(ii) other characters, except %, which are expected to match the next non-white space character in input.
(iii) conversion specifications, which the following form:
%*wlc

*, w and f are optional.

* causes 'assignment suppression'. The corresponding input field will be skipped without storing the result. There should not be a corresponding pointer argument.

w (width) is a decimal number specifying the maximum number of characters in the input field.

l indicates that the argument is a pointer to a long, if the conversion character is d, i, o, u or x.

c (conversion character) is one of the following:

o octal integer. The argument must be a pointer to an int.

x hexadecimal integer. The argument must be a pointer to an int.

u unsigned integer. The argument must be a pointer to an unsigned.

d integer. The argument pointer must be a pointer to an int.

i integer. If it starts with 0x or 0X, it is taken to be hexadecimal. Otherwise if it starts with a 0, it is taken to be octal.The argument must be a pointer to an int.

e floating point number. The argument must be a pointer to a double.

f same as e.

c single character. The argument must be a pointer to a char. This suppresses the skipping of whitespace characters. To read the next non-white space character, use %1s.

s string. The argument must be a pointer to a char array, large enough to hold the string (plus the terminating zero byte).

[string with specified acceptable characters. Following the '[' is a string of acceptable characters, terminated by ']'. If the first character after '[' is '↑' the characters which follow are considered not acceptable. The argument must be a pointer to a char array, large enough to hold the string (plus the terminating zero byte).

Returns: the number of items that were assigned, or *EOF* (-1) if the end of file is reached, or an error occurs before the end of the format string is reached.

seldrive
int seldrive(dr)
int dr;

Description: Arnor C

Makes drive *dr* the currently selected drive.

Returns: 0 if successful, *ERROR* (-1) if an error occurred.

selwin
void selwin(window)
int window;

Description: Arnor C, Small

Selects the specified window number, which is a number between 0 and 7. If *setwin* has not been previously used with the specified window selected, the window will cover the whole screen. The cursor position is retained separately for each window.

Returns: nothing.

setcurs
void setcurs(col,row)
int col,row ;

Description: Arnor C, Small

Sets the cursor position to the specified column and row within the current window. The top left of the window is position (0,0).

Returns: nothing.

setfcb
void setfcb(name,buf)
*char *name ;*
char buf[36] ;

Description: Arnor C, Small

This takes a filename and constructs a file control block ready for use with BDOS file routines.

Returns: nothing.

setjmp
int setjmp(env)
jmp_buf env ;

Description: Standard, Small

Allows a 'goto' between functions. *setjmp()* saves the stack environment in the variable *env* for later use by *longjmp()*. *setjmp()* must be called with a particular value of *env* before that value of *env* is used with *longjmp()* and the function in which *setjmp()* is called must not have returned before the corresponding *longjmp()* is used.

Returns: zero.

setpr
void setpr(printer)
int printer;

Description: Arnor C

Selects destination for *prch*. *printer* is a number representing the required printer output, as follows:

1 PCW internal printer
2 Serial printer
4 Parallel printer

Returns: nothing.

settime
void settime(tod)
*timeblock *tod;*

Description: Arnor C

Sets the system clock (using BDOS function 104). *tod* is a pointer to a structure defined as follows:

```
typedef struct
{
    int date;       /* in days since January 1, 1978 */
    char hour;
    char minute;
    char second;
} timeblock;
```

Returns: nothing.

setwin
void setwin(x1,y1,x2,y2)
int x1,y1,x2,y2;

Description: Arnor C, Small

Sets the window so that top left is (*x1,y1*), and bottom right is (*x2,y2*). The co-ordinates used are absolute physical screen co-ordinates, top left of the screen is position (0,0). The cursor is moved to the top left of the window.

Returns: nothing.

sin
double sin(x)
double x;

Description: Standard, Maths

Calculates the sine of the argument which is in radians.

Returns: the sine of the argument which is in radians.

sinh
double sinh(x)
double x;

Description: Common, Maths

Calculates the hyperbolic sine of the argument.

Returns: the hyperbolic sine of the argument.

sprintf
int sprintf(buff,form,args...)
*char *form,*buff;*

Description: Standard

Writes a string to the memory buffer *buff* using the format string *form* and inserting the arguments accordingly. See *printf* for further information on how the format string is interpreted.

Returns: 0 if successful, *ERROR* (-1) if unsuccessful.

sqrt
double sqrt(x)
double x;

Description: Standard, Maths

Calculates the square root of the argument, which must be non negative.

Returns: the square root of the argument.

srand
void srand(seed)
unsigned int seed ;

Description: Standard, Maths

Seeds the random number generator with *seed*. The initial default seed is 1.

Returns: nothing.

sscanf
int sscanf(buff,form,args...)
*char *form,*buff ;*

Description: Standard

Reads a string from the buffer *buff* using the format string *form*, extracting the arguments accordingly and storing them through the *args* pointers. See *scanf* for information on how the string is interpreted.

Returns: the number of arguments extracted.

strcat
*char *strcat(s1,s2)*
*char *s1,*s2 ;*

Description: Standard, Small

Appends *s2* to *s1*, terminating the result with a zero.

Returns: a pointer to *s1*.

strchr
*char *strchr(s,c)*
*char *s ;*
char c ;

Description: Common, Small

strchr searches the string *s* for the first occurrence of the character *c*.

Returns: a pointer to the character, or *NULL* (0) if not found.

strcmp
int strcmp(s1,s2)
*char *s1,*s2;*

Description: Standard, Small

Compares the strings *s1* and *s2*.

Returns: 0, if equal, a negative value if *s1*<*s2*, or a positive value if *s1*>*s2*.

strcmpl
int strcmpl(s1,s2)
*char *s1,*s2;*

Description: Common

Compares the strings *s1* and *s2*, ignoring differences between upper and lower case letters.

Returns: 0, if equal, a negative value if *s1*<*s2*, or a positive value if *s1*>*s2*.

strcpy
*char *strcpy(s1,s2)*
*char *s1,*s2;*

Description: Standard, Small

Copies the string *s2* to *s1*.

Returns: a pointer to *s1*.

strcspn
int strcspn(s1,s2)
*char *s1,*s2;*

Description: Common

Evaluates the number of consecutive characters at the start of *s1* that are not contained within the string *s2*. See also *strspn*.

Returns: the number of consecutive characters at the start of *s1* that are not contained within the string *s2*.

strdup
*char *strdup(s1)*
*char *s1;*

Description: Common

Allocates memory for a copy of *s1* using *malloc* and copies the string there.

Returns: a pointer to the copy.

strlen
int strlen(s1)
*char *s1;*

Description: Standard, Small

Evaluates the number of characters in *s1* excluding the terminating zero.

Returns: the number of characters in *s1* excluding the terminating zero.

strlwr
*char *strlwr(s1)*
*char *s1;*

Description: Common

Converts all upper case letters in *s1* to lower case.

Returns: *s1*

strncat
*char *strncat(s1,s2,n)*
*char *s1, *s2 ;*
int n ;

Description: Standard, Small

Appends up to *n* characters from *s2* to *s1*, terminating the result with a zero.

Returns: a pointer to *s1*.

strncmp
int strncmp(s1,s2,n)
*char *s1, *s2 ;*
int n ;

Description: Standard

Compares up to *n* characters of *s1* with *s2*.

Returns: 0, if equal, a negative value if *s1*<*s2*, or a positive value if *s1*>*s2*.

strncpy
*char *strncpy(s1,s2,n)*
*char *s1, *s2 ;*
int n ;

Description: Standard, Small

Copies up to *n* characters from *s2* to *s1*. If *strlen(s2)* > = *n*, *s1* will not be zero terminated.

Returns: a pointer to *s1*.

strpbrk
*char *strpbrk(s1,s2)*
*char *s1, *s2 ;*

Description: Common

Finds the first character in *s1* that is in the string *s2*.

Returns: a pointer to the first character in *s1* that is in the string *s2*.

strrchr
*char *strrchr(s1,c)*
*char c,*s1;*

Description: Common

Finds the last occurrence of the character *c* in *s1*.

Returns: a pointer to the last occurrence of the character *c* in *s1*.

strrev
*char *strrev(s1)*
*char *s1*

Description: Common

Reverses the order of characters in *s1*.

Returns: a pointer to *s1*.

strset
*char *strset(s1,c)*
*char c,*s1;*

Description: Common

Sets all characters in *s1* to the value of *c*.

Returns: a pointer to *s1*.

strspn
int strspn(s1,s2)
*char *s1,*s2;*

Description: Common

Evaluates the number of consecutive characters at the start of *s1* that are contained within the string *s2*. See also *strcspn*.

Returns: the number of consecutive characters at the start of *s1* that are contained within the string *s2*. A value of 0 indicates that no substring was found.

strtod
double strtod(str,ptr)
*char *str, **ptr ;*

Description: Common, Small

Converts the string *str* to a floating point number. Conversion stops at the first inappropriate character, and if *ptr* is not *NULL* (0) then **ptr* is set to point to this character.

Returns: if no number is found, 0.0 is returned.

strtok
*char *strtok(s1,s2)*
*char *s1, *s2 ;*

Description: Common

Looks for the next token in the string *s1*. A token is a sequence of characters separated by one or more delimiter characters. The delimiter characters are those contained in the string *s2*.
To find subsequent tokens, *strtok* must be called with *NULL* (0) as the first argument. This will cause the search to resume after the previous token.

Note: The contents of *s1* are changed.

Returns: a pointer to the start of the token (which will be zero terminated), or if no token is found, *NULL* (0).

strtol
long strtol(str,ptr,base)
*char *str, **ptr ;*
int base ;

Description: Common, Small

Converts the string *str* to a long integer. The string is assumed to hold the ASCII representation of a number in base *base*. If *base* is zero, the base is derived from the string: if it starts with 0x, base 16 is used, otherwise if it starts with 0, base 8 is used, otherwise base 10 is used. Conversion stops at the first inappropriate character, and if *ptr* is not *NULL* (0) then **ptr* is set to point to this character.

Returns: if no number is found, 0L is returned.

strupr
*char *strupr(s1)*
*char *s1;*

Description: Common

Converts all lower case letters in *s1* to upper case.

Returns: nothing.

tan
double tan(x)
double x;

Description: Common, Maths

Calculates the tangent of the argument, which is in radians.

Returns: the tangent of the argument.

tanh
double tanh(x)
double x;

Description: Common, Maths

Calculates the hyperbolic tangent of the argument.

Returns: the hyperbolic tangent of the argument.

time
void time(tod)
*timeblock *tod ;*

Description: Arnor C

tod is a pointer to a structure as defined in *settime*.

Returns: the time.

toascii
int toascii(c)
int c ;

Description: Common, Macro

Takes any integer value and discards all but the low order seven bits making up an ASCII character.

Returns: integer value of low order seven bits, or if already valid, returns unchanged.

tolower
int tolower(c)
int c ;

Description: Standard, Macro

Converts *c* to lower case if it was an upper case letter.

Returns: *c*, converted to lower case if it was an upper case letter, otherwise, unchanged.

toupper
int toupper(c)
int c ;

Description: Standard, Macro

Converts *c* to upper case if it was a lower case letter.

Returns: *c*, converted to upper case if it was a lower case letter, otherwise, unchanged.

ungetc

int ungetc(c,fp)
int c ;
*FILE *fp ;*

Description: Standard, Small

Puts the character *c* back into the input stream *fp*, where it will be read by the next input operation on that stream. Only one character may be pushed back between input operations.

Returns: *c* if successful, or *ERROR* (-1) if unsuccessful.

ungetch

void ungetch(c)
char c ;

Description: Common, Small

Returns a character to the console input.

Returns: *c*, if successful, otherwise *ERROR* (-1).

unlink

int unlink(name)
*char *name ;*

Description: Standard, Small

Deletes the specified file.

Returns: 0 if the file was successfully deleted or *ERROR* (-1) if an error occurred.

unwrchar

int unwrchar()

Description: Arnor C, Small

Attempts to read a character from the current cursor position.

Returns: character read, otherwise *ERROR* (-1).

version
unsigned version()

Description: Arnor C, Small

On a CPC6128 this returns the version number of the C run time system, multiplied by 100.
On a PCW it returns the same, with 32768 added. Thus to test whether a program is running on a PCW, use *if (version() & 0x8000)*.

Returns: the version number of the C run time system, multiplied by 100, on a CPC6128. On a PCW it returns the same, with 32768 added.

wrchar
void wrchar(c)
char c ;

Description: Arnor C, Small

Writes a character to the screen, even if *stdout* has been re-directed. All characters are displayed, control codes are not interpreted.

Returns: nothing.

write
int write(h,buf,n)
int h, n ;
*char *buf ;*

Description: Standard, Small

Writes *n* bytes from the buffer *buf* to the file with handle *h* and stores them in the buffer *buf*.

Returns: the number of bytes transferred, or *ERROR* (-1) if an error occurs.

wrmatrix
void wrmatrix(c,buf)
char c ;
char buf[8] ;

Description: Arnor C, Small

Sets the matrix for the character *c* as passed in the 8 byte buffer. The bytes passed are the bit image data for each row of the character matrix, starting with the top row. Within each byte, bit 7 is the leftmost pixel.

Returns: nothing.

_exec
void _exec(str)
*char *str ;*

Description: Standard, Small

Takes a pointer to a string, and passes it to the run time system command line interpreter.

Returns: never returns and is used for chaining other programs.

_exit
void _exit(ecode)
int ecode ;

Description: Standard, Small

Does not flush buffers but returns immediately from the program. *ecode* is normally a zero to indicate a normal end of program and non-zero to indicate an error.

Returns: doesn't return.

_fchret

int _fchret(c,handle)
char c ;
int handle ;

Description: Arnor C, Small

Puts back the character *c* to the file.

Returns: 0 if successful, *ERROR* (-1) if an error occurred.

_feof

int _feof(handle)
int handle ;

Description: Arnor C, Small

Tests whether end of file has been reached.

Returns: 0 if not end of file. *0x1A* is returned if at soft end of file, a different non-zero number if at hard end of file.

_finch

int _finch(handle)
int handle ;

Description: Arnor C, Small

Reads a character from the file. Programs may need to check for *0x1A*, the soft end of file character.

Returns: *EOF* (-1), if hard end of file reached, otherwise the character read.

_foutch

int _foutch(c,handle)
char c ;
int handle ;

Description: Arnor C, Small

Writes a character to the file.

Returns: 0 if successful, *ERROR* (-1) if an error occurred.

_ftell

long _ftell(handle)
int handle ;

Description: Arnor C, Small

Evaluates the current file position as a long integer.

Returns: the current file position as a long integer.

_getlim

*char * _getlim*

Description: Arnor C, Small

Evaluates the lowest free memory address. See *_getsp*.

Returns: the lowest free memory address.

_getsp

*char * _getsp()*

Description: Arnor C, Small

Returns the highest free memory address (below the lowest extent of the stack). This routine is provided for use of dynamic memory allocation functions, such as *malloc* and will not normally be required.

Returns: the highest free memory address.

_putlim

void _putlim(address)
*char *address ;*

Description: Arnor C, Small

Sets the lowest free memory address. This function should only be used to raise the memory limit and then only if absolutely necessary (such as in a function like *malloc*).

Returns: nothing.

10. THE ARNOR C IMPLEMENTATION

This section gives implementation specific information, and documents any variations from or enhancements to the Kernighan & Ritchie standard. The section numbers correspond to those in the "C Reference Manual" section in K & R.

1. Introduction

This implementation is for CP/M Plus running on the Amstrad PCW8256/8512 and CPC6128.

2.1 Comments

Comments may not be nested.

2.2 Identifiers

Identifiers may be any length, but only the first eight characters are significant. The first character must be a letter or the underline character, but the remaining characters may be letters, numbers or the underline character '_'. Spaces are not permitted. Upper and lower case are treated as being different.

Identifiers are split up into the following groups, between which there are no possible clashes.

(i) Labels
(ii) Elements of structures / unions
(iii) Global variables
Typedef names
Structure / union tags
(iv) Local variables / parameters

In the case of local variables, a declaration of a particular variable causes any previous uses of that identifier to become inaccessible until the scope of the local variable expires.

2.3 Keywords

auto	double	int	struct
break	else	long	switch
case	extern	register	typedef
char	float	return	union
continue	for	short	unsigned
default	goto	sizeof	void
do	if	static	while

2.4.3 Character constants

The following character constants are provided:

(a) Mentioned in K & R

\n	newline (stored as 10)
\t	tab
\b	backspace
\r	carriage return
\f	form feed
\\	backslash
\'	single quote
\ddd	octal constant
\0	null character

(b) Additional

\xhh	hexadecimal constant
\c	conditional newline (only if not at start of line)
\v	vertical tab (reverse line feed)
\"	double quote

2.6 Hardware characteristics

See section 8.2 (type specifiers).

7 Expressions

Division by zero causes a run time system error, and execution of the program will be terminated. Floating point exceptions are handled by the library function 'matherr'.

8.1 Storage class specifiers

Register is recognised and treated as auto.

8.2 Type specifiers

The following combinations of type specifiers are allowed (brackets denote optional clauses) :-

type	storage	range (approx.)
char	1 byte	0 to 255
int	2 bytes	-32768 to 32767
short (int)	2 bytes	-32768 to 32767
long (int)	4 bytes	-(2↑31) to 2↑31-1
unsigned (int)	2 bytes	0 to 65535
(long) float	5 bytes	-1e38 to +1e38
double	5 bytes	-1e38 to +1e38
struct	-	
union	-	
<typename>	-	

<typename> refers to a user defined type; typedef is fully implemented.

The provision of a type specifier is optional in a global declaration or a local declaration where no ambiguity is possible. The default type is integer. Types double and float have the same precision.

8.5 Structure and union declarations

Bitfields are not implemented.

8.6 Initialisation

Initialisation of globals/locals is as described in K&R. Attention is drawn to what constitutes a 'constant expression' in the case of global initialisation, the compiler must be able to evaluate the constant expression to a (possibly relocatable) value at compile time of that module. Consider the following global initialisation:-

```
int ptrdiff = &a - &b;
```

This is illegal if either a or b is external - since their addresses cannot be determined - but is valid if a and b are defined in the same module as the declaration of ptrdiff.

Initialisation of multi-dimensional arrays is extended so that more than one array bound may be omitted. For example the declaration :-

```
int arr[][] = { {1,2,3} , {4,5,6} , {7} , {} };
```

is allowed, and is taken to declare an initialised array of size arr[4][3].

10 External definitions

References to functions that have not yet been defined or declared are taken to be references to external functions returning integers. Thus if a function is of storage class static, or it returns a value other than integer, it must be defined, or at least declared, before it is referenced in a module. In case of static functions the function must be defined before use. If this is not done the function will be coerced to be extern.

12.2 File inclusion

#include "filename" looks ONLY on the current drive
#include <filename> looks on ALL drives

Limits

arguments in command tail	35
number of files open	16
number of streams open	20 (if standard streams unchanged)
line length	300
length of string literal	32767
number of macro arguments	10

THE EDITOR

1. INTRODUCTION

The Arnor Program Editor is a full implementation of the program mode of the PROTEXT word processor and provides, amongst many other advanced features, facilities to edit two files at the same time and to transfer text between the two files.

Facilities are provided to copy, rename and delete files, as well as to format and back up discs, all without having to leave the editor, or lose the contents of any files in memory.

Screen output may be spooled to a file on disc, or redirected to the printer if required. Another special feature of the editor is its ability to carry out sequences of commands by means of EXEC files. These are special text files which may contain text, commands, or a combination of both, which will be treated as if they were being typed in at the keyboard.

Utility programs are provided to enable the editor to be configured to suit individual users preferences and these settings may be saved and will be used automatically on all future occasions if required.

Listed below is a summary of the conventions used in the manual to describe the various types of commands used in the editor:

a). Edit mode commands

ALT-M — means the key marked 'ALT' and the key marked 'M'. Wherever a hyphen is used between them, it means that the first key should be held down whilst the second key is pressed. Most of the editing commands take this form.

ALT-V T — means that the 'ALT' and 'V' keys should be used as described above, then released and the 'T' key pressed. Note that there is no hyphen between the 'V' and the 'T'.

ALT-SHIFT-H — means that all three keys should be pressed at the same time. This sort of command, which requires more than two keys to be pressed at a time is rarely used and at least two of the keys are always adjacent to each other.

ALT-@ — means that the 'ALT' key and the key which has the '@' on it are pressed together. It does NOT mean that SHIFT is required as well. The '@' is merely being used for ease of remembering its function.

b). Command mode commands

Command mode commands are always shown in upper case, though when they are being entered into the computer, they may equally well be entered in upper or lower case. Similarly, when entering filenames to LOAD or SAVE a file, even though they may be shown in upper case, lower case is acceptable and the editor will automatically convert them to upper case if required.

c). Key variations for the CPC6128

There are a number of differences between the keyboards of the PCW computers and the CPC computer. Throughout this section of the manual PCW key names are the ones which are used, rather than cause confusion by listing both keys on all occasions.

The following keys are direct equivalents:

PCW key	CPC key
STOP	ESC
EXIT	ESC
ALT	CONTROL
DEL→	CLR
←DEL	DEL

There is no direct equivalent to the EXTRA key, but for most purposes, CTRL-0 (zero) serves the same purpose.

Any other variations, where, for example, there is no directly equivalent key on the CPC6128, are noted at the relevant places.

2. EDIT MODE

Throughout this part of the manual, the standard editor command keys are described. The PCW computers have a number of 'special' keys on the right hand side of the keyboard and these have been configured to correspond to their originally intended uses as far as possible, which in most cases is merely a duplication of the equivalent editor command. It should, however, be noted that due to the different methods used by the editor for moving, copying and deleting blocks of text, there is some variation in the way that the CUT, CAN and COPY keys are used.

A complete summary of both types of command is given in an appendix.

Note: The 'NUM LOCK' facility provided with the PCW computer, obtained by pressing ALT-RELAY, is not available with the editor. It has been disabled, as accidental use of the option, which changes the use of the cursor and other keys into numeric keys, can be very confusing, as the cursor keys would no longer function as expected.

Full use is made of the cursor keys during editing and when used in conjunction with SHIFT, or ALT, the effect becomes increasingly greater. For example: Using the right cursor key on its own will move the cursor one character at a time. Using it with the SHIFT key will move a word at a time, whilst with ALT, it will move to the end of the line.

Similarly, the commands to delete make use of the two DEL keys, which on their own will delete one character, but when used with SHIFT will delete a word and with ALT will delete to the beginning or end of the line.

a). Editing

Once the editor has been loaded, two lines containing information about the state of the program will be seen at the top of the screen. These are the 'Status lines', the contents of which will be explained later. There is also a thin horizontal line, which always marks the end of the text and about two thirds of the way down the screen is another, broader, line containing further information.

At this stage the program is still in Command Mode, which is described in detail in the next chapter, but pressing the STOP key will put the program into Edit Mode, which is the mode used for all entry and correction of text. The line two thirds of the way down the screen will disappear, leaving the lower part of the screen clear. Pressing the STOP key at any time will return to command mode.

b). On screen help

There are two distinct types of 'on screen help' available. One is for help whilst in edit mode and the other for help in command mode. Full details on the Command mode help are given in the chapter on 'Command mode'.

At the top of the screen, on the status line, will be found the message 'ALT-H for Help'. Pressing ALT-H at any time whilst in edit mode will turn on a help panel at the bottom of the screen. If required this may be left permanently on, but once some degree of familiarity with the editor has been attained it would normally only be turned on when information about a particular command was required.

Pressing ALT-H subsequently will display further panels of help information. When the last panel is reached, the display will repeat from the start again. Pressing ALT-V B will move backwards through the help panels.

Pressing ALT-V H will turn the help panel off at any time and restore editing to the full screen.

Note: In order for help to be available, the files called AEDIT.HLP and ACOMMAND.HLP must be available on one or other of the drives. In the case of the PCW, they would normally be present on drive M, having automatically been transferred to drive M as part of the 'Start up' procedure. CPC6128 users would normally need to have copies of these files on their text disc, or the start of day disc in one of the drives.

c). Entering text

Once in edit mode a flashing cursor is positioned beneath the status lines and anything that is typed at the keyboard will appear on the screen at this position and the cursor will be moved forward one position.

Any mistakes made whilst typing, which are noticed at the time, may be corrected by pressing the ←DEL key, which will cancel the last key pressed.

The cursor can be moved around the screen by pressing the four cursor keys. By using these keys, text may be entered at any position. The cursor moves one line or column for each press of a cursor key. Holding a cursor key down will make the cursor move continuously - release the key and the cursor will stop.

The cursor cannot be moved past the end of text (the thin horizontal line on the screen). To position the cursor further down, the end of text must be moved down by positioning the cursor at the end of the text and pressing RETURN as many times as required.

d). Upper and lower case

Initially the letter keys produce lower case letters, unless SHIFT is pressed at the same time. If SHIFT LOCK or CAPS LOCK is pressed, upper case letters are always produced, and this is indicated on the status line.

Note: The Amstrad PCW computers are slightly unusual in having a SHIFT LOCK and no CAPS LOCK key. When SHIFT LOCK is on, all the characters on the upper part of those keys which have more than one character on them will also be selected. Caps lock is selected by pressing ALT-ENTER.

Note: On the CPC6128, SHIFT LOCK may be obtained by pressing CONTROL and CAPS LOCK together.

The editor has two commands which change the case of a letter. To make a letter upper case, press ALT-/ when the cursor is on the letter. This command only affects letters, so the cursor can be moved quickly over a line to convert all letters to upper case by holding down ALT-/. Similarly, ALT-. (point) will convert upper case letters into lower case. (CPC6128 equivalent: CTRL-\).

e). Deleting and inserting

The ability to move the cursor around, permits the correction or alteration of text anywhere on the screen. The cursor should be positioned on the letter to be changed and the DEL→ key pressed. This will remove the letter at the cursor position, and move the rest of the line to the left. As many letters as required can be deleted in this way. If the new letter is now entered it will appear on the screen and the rest of the line will move back to the right. Alternatively, pressing ←DEL will remove the character to the left of the cursor and the text will again move to the left to fill the gap. Repeated pressing of either DEL key will cause further characters to be deleted.

If extra text is to be inserted, the cursor should be positioned where the first new character is to be added and the new text entered.

To insert a new blank line into the text, ALT-I should be used. The cursor will remain where it is and all text from the current line to the end of the document will be moved down a line.

Just as a character can be deleted, so can a word. Pressing SHIFT and DEL→ when the cursor is at the start of a word will make the word disappear. If this is done when the cursor is in the middle of a word, only that part of the word at and to the right of the cursor position will be deleted.

Similarly, pressing SHIFT and ←DEL will remove the word to the left of the cursor, or if positioned in the middle of a word, the characters to the start of the word.

ALT-←DEL will delete all text from the character on the left of the cursor to the start of the line and ALT-DEL→ will delete all text from the cursor to the end of the line. ALT-E also deletes everything from the cursor position to the end of the line. (CPC6128 users: only CTRL-E is available).

ALT-CAN will delete the whole line. The line is removed from the document and the remainder of the text moved up a line. (CPC6128 equivalent: CTRL-CLR).

Note: On the PCW, pressing ALT-←DEL followed by ALT-DEL→ will delete all the text from a line, but will not remove the empty line from the text, unlike ALT-CAN, which will remove the blank line as well.

f). Swapping two characters

A common typing mistake, especially when typing quickly, is to type two letters the wrong way round, e.g. 'wrod' instead of 'word'. The ALT-A (Alternate characters) command will put this right. The cursor should be positioned on the first of the two offending characters (on the 'r', in the above example) and ALT-A pressed. The two characters will then be exchanged.

g). Un-deleting all or part of a line

The editor maintains a buffer which always contains the most recently deleted section of text. If a line or part of a line, more than three characters long, is deleted, the deleted text will be saved in the buffer. If a section of text has been accidentally deleted, it may be restored by pressing ALT-U.

This command can also be put to good use for moving lines or parts of a line to a different position in the text, though this is not the purpose for which it is really intended. The text to be moved should be deleted using one of the word or line delete commands and the cursor moved to the position in the text where the deleted text is to be placed. Pressing ALT-U will then restore the text at the new location.

Note: Only the text removed by the last delete command will be stored in the buffer and any previous contents of the buffer will be lost. It is therefore only possible to un-delete a section of text until such time as any other section of text is deleted.

h). Insert and Overwrite mode

Initially the editor, by default, is in insert mode and the word 'Insert' is displayed on the status line at the top of the screen to indicate this. This means that when text is typed, the rest of the text on the line is moved along to the right to make room. This is the mode that is preferred by most people for text entry.

Pressing ALT-TAB will change the status line to 'Overwrite'. Selecting overwrite mode can make certain editing jobs easier. The effect of using it is that if the cursor is positioned over an existing piece of text and new text typed in, the existing text will be replaced by the new text, unlike insert mode, where the existing text would be moved to the right.

If an extra character needs to be inserted whilst in overwrite mode (for example if replacing a word by a longer word), this can be done by pressing ALT and the space bar which will move the text to the right to make room.

i). Moving the cursor more rapidly

So far the cursor has been moved by a character at a time, but there are also various ways to move the cursor more quickly. These are as follows:

(a) Pressing SHIFT-→ or SHIFT-← will make the cursor jump a word to the start of the next (or last) word. This feature is useful for moving more quickly to a word which needs correction.

(b) Pressing ALT-← or ALT-→. This moves the cursor to the beginning or end of the line.

(c) Pressing SHIFT-RETURN or ALT-RETURN. This moves the cursor to the beginning of the next line, without causing a new line to be inserted, which would happen if the RETURN key was used on its own.

(d) Pressing ALT-↑ or ALT-↓. This moves the cursor up or down rapidly. By holding down ALT-↑ or ALT-↓ the text can be rapidly scanned. The text will scroll by nearly a screenful at a time, but with a few lines overlap so that the context may more easily be followed. Similar functions are performed by ALT-Q and ALT-Z, except that a full screen is scrolled each time, with no overlap of text.

(e) Pressing ALT-[or ALT-] moves the cursor to the beginning or end of the text resident in memory at that time. Pressing the same key a second time will move the cursor to the beginning or end of the complete document.

(f) Pressing ALT-@ [or ALT-@] will move to the opening or closing block markers, if set. See 'Block commands'.

(g) Pressing ALT-⊞ or ALT-⊟ will go to the next or previous marker in the document. See 'Place markers'.

Note: The ⊞ and ⊟ keys are the special plus and minus keys located either side of the space bar on the PCW. CPC6128 users should use CTRL-@ + and CTRL-@ -.

(h) Pressing ALT-L moves the cursor back to the last position. This is particularly useful if the cursor has accidentally been moved to another part of the text by using an incorrect command. ALT-L will return the cursor to the position where it was before the incorrect move was made. It will only have any effect if the cursor has been moved with one of the 'jump' commands. Moving the cursor a single space at a time will not affect the use of ALT-L and it can still be used to return to the original position from which the last jump was made.

With care, this facility can be put to good use, by permitting a jump to another part of the text, where one or two alterations or additions may be made, before pressing ALT-L to return to the original place in the text.

j). Moving to a specified line or column number

Pressing ALT-G will result in a message appearing on the status line, requesting 'L(ine) or C(olumn) number'. Entering the required line number will move the cursor to that line. Prefixing a number with 'C' will result in the cursor moving to the appropriate column.

k). Place markers

A place marker can be put anywhere in the text and is similar in use to a book marker. Ten place markers can be set, numbered 0 to 9. A place marker is set by pressing ALT-@ followed by the number. When a marker has been set, it will appear in the text as the number in inverse and will be shown on the status line, so that by looking at the status line it is easy to see which markers are available. Once a place marker has been set, it can easily be returned to at any time by repeating the ALT-@ command with the same number.

In addition to using ALT-@ and the number to find a place marker, it is possible to jump from one to the next in the document by using ALT-⊞ to move on through the document, or ALT-⊟ to move backwards. Using these commands will find the next or last marker in the text. Both types of markers (place, and block) will be found. They are not treated numerically, but are found in the order in which they occur in the document. (CPC6128 equivalents: CTRL-@ + moves to next marker, CTRL-@ - moves to previous marker).

As an example of the use of a place marker, suppose a long file is being edited and something needs to be added at the top of the text. A place marker can be set and ALT-[typed, to move to the top of the text, and after making the addition, ALT-@ and the place marker number used to move back to the place marker.

Note: Place markers are saved with the text and will be restored when the file is reloaded. If one file containing markers is merged into another, duplication of markers may occur. The duplicated markers can be removed in the normal way.

l). Scrolling

When the text fills the entire depth of the screen, typing further text will cause the screen to scroll up. That is, the top line will disappear and the rest of the screen will move up one line to make room for a new line at the bottom of the screen.

In the same way the text will scroll if the cursor reaches the bottom of the screen but there is more text to come, or reaches the top of the screen when the text has previously scrolled. This is known as vertical scrolling, and is essential for editing text that is longer than a few lines.

The editor has commands to force the screen to scroll either up or down at any time. This is done by pressing SHIFT-↑ or SHIFT-↓. The cursor will stay on the same line, but the whole text will scroll by one line. This feature is useful if a line is to be edited and it is desirable to see the text beneath or above.

There is another form of scrolling, called horizontal scrolling, which happens automatically when the cursor is moved beyond the right hand limit of the screen. If this is done the text will scroll to the left. This means that the text on the left of the screen will start to disappear as the cursor is moved further to the right of the screen. Horizontal scrolling allows text to be entered in lines that are longer than the screen width. This can be confusing at first and so is best avoided initially. If horizontal scrolling occurs, any of the commands which move the cursor to the left may be used to scroll the text back, or SHIFT and RETURN may be pressed together, which will return the cursor to the start of the next line.

Note: When horizontal scrolling takes place, the screen moves across a number of spaces at a time. This figure may be changed with the CONFIG utility but it should be borne in mind that the smaller the movement, the more it will slow down the operation of the editor.

m). Splitting and joining lines

Lines will often require splitting, or joining together. This is very easy in the editor. There are two different methods of doing this, depending on whether 'Insert' or 'Overwrite' mode is in operation.

To split a line whilst in Insert mode, the cursor should be moved to the character which is to be the first on the new line and RETURN pressed. To join two lines, either move to the end of the first line and press DEL→, or move to the start of the second line and press ←DEL. The text on the second line will then move up and join onto the end of the text on the first line.

If in overwrite mode, ALT-* will split the line at the cursor and ALT- + will join the next line to the end of the current line.

n). Tabs

The TAB key may be used at any time to indent text and align columns of figures or text. Use of the TAB key to position characters, rather than inserting spaces in the text, is recommended for a number of reasons.

Firstly, it provides a quick and efficient method of neatly presenting source code, but another benefit is that a single tab in the text can take the place of a number of space characters, making the resultant file shorter.

Secondly, it is possible to rearrange the spacing of columns at any time, simply by re-specifying the tab spacing required.

Tab markers are set at every eight columns by default, but this may be changed at any time by use of the command, 'TAB', from command mode. This command is fully described in the chapter on command mode. One thing which should be remembered is that any special tab settings will be lost when the document is saved and next time it is loaded, the settings will be the default ones again.

If it is felt desirable to always use a special set of tab spacings, then these may be saved and automatically be loaded by use of the AUTOEXEC file feature of the editor. Full details of how to create a suitable EXEC file are given in the section on 'EXEC files'.

3. BLOCK COMMANDS

a). Block commands

The editor allows any section of text to be moved or copied to any other part of the text. This is often called 'cut and paste' editing. A block of text is any continuous section of text. It may be of any length and may start at any position in the document and finish at any position. When in block editing mode, all text between these two points will be manipulated in whatever way is chosen.

b). Defining a block

The first requirement is that the block of text is marked with block markers. The cursor should be moved to the start of the section of text and SHIFT-COPY pressed (alternatively SHIFT-⊞ can be used and may be found more convenient). This will set a block marker. The marker will be indicated on the screen by an inverse video square bracket. The cursor should then be moved to the end of the section and SHIFT-COPY pressed again, to set a second marker. The block has now been defined. An opening square bracket is the start marker, a closing square bracket the end marker. When markers are defined, this will be indicated on the status line, where the message 'No markers set' will be replaced by 'Markers [ ]', showing that both the start and end markers are set.

The markers can be set in either order, and can be at any position in the text. The first marker set will be displayed as an opening bracket, but if the second marker is positioned earlier in the text than the first marker, this will change to a closing bracket. If the marker is put in the wrong place, pressing SHIFT-COPY again while the cursor is still on the marker will remove it. Either or both block markers can be cleared at any time, by pressing ALT-K or CAN. Often a block will consist of a number of complete lines. To define a block like this, the first marker should be positioned at the start of the first line, and the second marker at the start of the line following the last line of the block.

If an attempt is made to set a marker when both are already set, a beep will sound and an error message will be displayed on the status line. Pressing STOP will resume editing and ALT-K can be used to clear the markers.

c). Moving or copying a block

Once a block has been defined, it can be moved to any point in the text simply by moving the cursor to the required position and pressing ALT-M or alternatively on the PCW, the PASTE key. The markers will move with the text. The cursor must not be within the block at the time; if it is an error message will be displayed on the status line. Pressing STOP will return to edit mode and the cursor can be moved to the correct position.

The block can also be copied, leaving the original text intact. This is done by pressing ALT-COPY or just COPY (PCW only). The markers will be moved with the block, which makes it easy to see clearly where the new copy of the block is and also to copy the block again if required. The cursor must not be within the block. (CPC6128 equivalent: CTRL-COPY).

d). Deleting a block

The section of text to be deleted must be defined in the usual way. Pressing the CUT key will delete the block. If the block is larger than a certain size (see below) a beep will sound and a warning message will be displayed on the status line, requesting confirmation that the block is to be deleted. The block will only be deleted if 'Y' is selected. (CPC6128 equivalent: CTRL-DEL).

e). Un-deleting a block

If a block of text is accidentally deleted, it may often be recovered by use of the ALT-U command. When text is deleted, the editor retains the deleted block in a buffer and ALT-U will restore it to the document.

Note: By default the buffer will hold 512 characters, but this can be changed by the user through the use of the configuration program CONFIG.COM. If a block of text which is too large for the buffer is to be deleted, a warning will be given, with the option to continue. If 'Y' is selected, the block will be deleted and the buffer will be filled with as much of the text as it can hold and the remainder will be discarded.

Note: A block can only be restored until such time as further text is deleted, after which time the buffer will contain only the most recently deleted text.

4. FIND AND REPLACE

Note: CPC6128 users should note that there is some variation in the use of keys in this section as the CPC6128 does not have the special ⊞ and ⊟ keys. The COPY key serves the same purpose as the ⊞ key and CTRL-@ @ is the substitute for ⊟. SHIFT-f2 and CTRL-f2 generate REPLACE and FIND respectively instead of the PCW EXCH and FIND keys.

Two functions, FIND and REPLACE are provided, which permit searching through text for any string of characters and, if specified, replacing them with a second string.

Pressing FIND or EXCH whilst in edit mode will cause the editor to enter command mode, with a request for the 'String to find'. Alternatively, typing FIND or REPLACE from command mode will have the same result.

The string to find is requested first, followed by the replacement string (if the REPLACE option was selected). After entering the string or strings, one or more of a number of options may be selected by typing the appropriate letters one after another (in any order). Each option is either a single letter abbreviation or a number (these are listed on the screen). Pressing RETURN on its own will cause no options to be selected.

The options available are as follows:

G Global search. If selected the whole text is searched from the start, otherwise only the text from the current cursor position to the end of the text.

C Case specific search. If selected all letters will only match letters that are the same case, otherwise either capitals or lower case letters will be treated as being the same.

W Find string only if it appears as a complete word. For example to find occurrences of the word 'and' without finding 'hand', 'England' etc.

B Search backwards. Searches from the end of the document to the beginning.

A Find or replace all strings automatically. REPLACE will change all occurrences of the string with the new one, without requesting confirmation and return a figure of the total number of replacements made. In the case of FIND being used, it will simply return the total number of occurrences of the string.

n Find or replace the nth occurrence. n should be a number between 1 and 255. This option has a number of uses, but a simple example might be to check that every set of quotation marks has a matching closing set, in which case FIND would be used to find '"' and '2G' would be specified as options, to search globally for every second occurrence.

If no options are selected the search will be forwards, from the current cursor position to the first occurrence of the string, ignoring the case of letters, finding the string even if it occurs as part of a longer word, and asking for confirmation before replacing a string.

Any number of wildcards are allowed in the string. A wildcard is a character that matches any character in the text, except the return character. It is entered in the string by typing a question mark (?). A tab character may be entered simply by pressing the TAB key. It is displayed as a right pointing arrow.

There are various characters that cannot be entered directly, but that it may be useful to include in a search string. Provision has been made for including these in a string, by means of an 'escape character'. The 'escape character' (!) should be typed in, followed by a symbol, number or letter, as appropriate.

The full list of characters that are entered by this means is:

question mark	!?
exclamation mark	!!
return	!.
search for code	!<number>

a). Using FIND

Once the string and any options have been selected, edit mode is entered and the cursor placed on the first character of the first occurrence of the string. To find the next occurrence of the string, the ⊞ key, positioned to the left of the space bar, should be pressed. This need not be done immediately. Editing can be carried out first and when complete, the search may be continued by pressing ⊞. At any stage, ⊟ can be used to search back towards the beginning, if necessary.

As with other commands, FIND can be used by typing the string on the same line as the command name, followed by any options. Thus the command 'FIND word GWC' will search for the string 'word' from the start of the document, selecting only those occurrences where it is a complete word with all letters in the same case as specified. If no options are specified, the default options will be used.

If the A option is selected, the editor will return the total number of occurrences found, when the search is complete.

b). Using REPLACE

The cursor will be positioned on the first character of the string and a message, 'Replace (y/n)?',will be displayed on the status line. Pressing 'Y' will replace the string with the new one and the cursor will move to the next occurrence. Pressing 'N' will leave the string untouched and move the cursor to the next occurrence. Alternatively STOP may be pressed and normal editing resumed. At a later time, ⊞ may be pressed to resume the find and replace operation. Alternatively ⊟ may be used to resume the search in the reverse direction, which may be found useful if an occurrence of the string is passed over by pressing 'N' in error.

If option A is selected then all occurrences of the string are replaced without prompting and the program will remain in command mode. When complete a count of the total number of changes made will be displayed.

Examples

1. To find all occurrences of the word 'text' in lower case only, starting at the cursor position.

   FIND string: text
   Options: CW

2. To convert all occurrences of 'rom' or 'Rom' to 'ROM', asking for confirmation of each replacement.

   FIND string: rom
   REPLACE with: ROM
   Options: GW

3. To insert a blank line after each line.

   REPLACE !. !.!. AG

5. COMMAND MODE

a). Introduction

All entry of text is carried out in Edit mode, but in order to carry out operations such as saving, loading or printing, 'command' mode must be entered. This can be done at any stage of editing simply by pressing STOP. Pressing STOP a second time will return to edit mode.

When STOP is pressed, the bottom part of the screen will be cleared and the command mode banner line will appear, displaying the editor version number. The cursor will be positioned next to a '>' symbol. This symbol is the 'command prompt' and indicates that commands may be entered. The currently selected drive is indicated by the letter prefixing the > and if any 'group' other than group 0 is selected, this number will also be indicated.

The output of all commands will be displayed in this window at the bottom of the screen. Many commands produce more output than will fit in the window in which case the screen will automatically scroll as necessary.

(i) Command HELP

Command mode HELP is available at any time. Typing HELP will give a list of the available subjects, such as FILES, PRINT, DISC, and EXTERNAL. Typing HELP, followed by the subject will list all the commands relevant to the subject. For example: HELP DISC, (or H D) will list all disc utility commands.

(ii) Command entry

Before studying the individual commands in detail, there are a number of points connected with the entry of commands which are of general interest and are listed below.

The editor has a special feature which permits the entry of commands in a simplified fashion. For example, to save a text file it is only necessary to type 'SAVE' and the editor will prompt with 'SAVE filename:' and wait for entry of a name for the text file.

Alternatively, the parameters of a command may be entered on the same line as the command name, e.g. 'LOAD source', 'SAVE prog'. In this way the commands may be used without the prompts for the parameters appearing, which is often more convenient when familiar with the syntax of the commands.

Note: All commands which **require** a parameter will prompt for them if the command is used on its own. Commands which have optional parameters require these to be entered at the same time as the command.

The editor provides a sophisticated line editing facility which is in operation whenever commands are being typed in. If a mistake is made the cursor can be moved back and the mistake corrected in the same way as in edit mode. The following editing commands are available in command mode:

PCW8256/8512	CPC6128	
←	←	Move cursor left one character.
→	→	Move cursor right one character.
ALT-←	CTRL-←	Move to start of line.
ALT-→	CTRL-→	Move to end of line.
DEL→	CLR	Delete at cursor.
←DEL	DEL	Delete before cursor.
ALT-A	CTRL-A	Alternate characters.
ALT-←DEL		Delete to beginning of line.
ALT-DEL→	CTRL-E	Delete to end of line.
ALT-TAB	CTRL-TAB	Switch between insert and overwrite modes.
CAN	CLR	Clear screen (if cursor at start of a line)
STOP	ESC	Abandon entry of current command.

Pressing COPY or ⊞ when the cursor is at the start of a line recalls the last command line used that was 4 or more characters in length, and positions the cursor at the end of the command. This has a number of uses, such as carrying out multiple saves of the same file, or repeating a load command which failed because the wrong disc had been inserted. Short commands such as 'A', 'CAT', 'SW' do not affect the command recalled.

In addition, the editor provides a 'copy cursor' facility when in command mode. If the SHIFT key is held down and one of the cursor keys used, a second cursor will appear and move according to the cursor keys. If this is positioned over a piece of text, the SHIFT key released and the ⊞ or COPY key pressed, the characters which are underneath the 'copy cursor' will be copied down to where the original cursor is positioned. Text may be copied from anywhere on the screen. This is a very convenient method of recalling commands which have been used previously, but can not be recalled by the method described in the previous paragraph because subsequent commands have been issued.

(iii) Abbreviations

Many of the commands can be abbreviated. For example, there is no need to type 'LOAD' in full, typing 'L' will serve the same purpose. Similarly 'S' for 'SAVE' and 'P' for 'PRINT'. A full list of the commands, abbreviations and their parameter syntax, is given later in this chapter and in an appendix.

(iv) The current filename

After a file has been loaded, or once a piece of text has been saved, the name of the file will be displayed on the status line. This becomes the 'current filename' and is remembered by the editor until changed, either by saving with another name, by use of the NAME command, or by loading a new file. Once a file possesses a current filename the name may be omitted when saving a file. Entering the SAVE command, and just pressing RETURN when the prompt 'SAVE filename:' appears, will save the file with the current filename. Care must be taken to ensure that it is indeed the correct name, to avoid accidentally erasing something else. If SAVE is typed and the name displayed is incorrect it can be edited as described above ('Command entry').

b). Editor commands

This chapter gives full details of the commands available in command mode. Details of the syntax used and what the command does are given together with any optional extensions to the basic command.

Many of the commands allow the use of ambiguous filenames. An ambiguous filename is one which contains 'wildcards'. The editor has two types of wildcards, which may be used in the same way as with CP/M commands.

- ? may be used to mean 'any single character'.
- * may be used to indicate 'any number of characters'.

For example:

DATA?.TXT	Any filename beginning with 'DATA' and having one further character (which may be blank), with the suffix 'TXT'.
B*.*	Any filename beginning with 'B', of any length and any suffix.
.	Any file.

Note: Only one '*' may be used in each part of the filename and suffix.

(i) Text file handling

CLEAR

Description: Clears the text currently in memory. A request for confirmation is made before this is done. On the PCW computer the same effect is achieved by typing ALT-SHIFT-CAN in edit mode.

LOAD (L)

Syntax: LOAD <filename>

Description: A document will be loaded into memory from a disc file of the specified name. A warning message will be given if the text currently in memory has not been saved. Press 'Y' to confirm that this text is to be discarded.

Note: If only the command name is entered, the editor will prompt for a filename. Once loaded, the specified filename will become the 'current filename'

MERGE (MER)

Syntax: MERGE <filename>

Description: This is similar to LOAD but whereas LOAD clears any existing text from memory and then loads the file in, MERGE inserts the new file into the existing text at the current cursor position.

Note: Care should be taken to ensure that the cursor is in the required position before using this command.

Note: The current filename is NOT changed.

NAME (N)

Syntax: NAME <filename>

Description: Permits the name of the document in memory to be changed. The new name becomes the 'current filename'.

SAVE (S)

Syntax: SAVE <filename>

Description: The complete document in memory will be saved to a disc file with the name specified.

Note: If only the command name is entered, the editor will prompt for a filename. If the file already has a 'current filename', then pressing RETURN will result in the file being saved with the same name. Alternatively, a new name may be specified, which will then become the current filename.

SAVEB (SB)

Syntax: SB <filename>

Description: This is the same as SAVE except that only the text within the block defined by the block markers is saved.

Note: The current filename is NOT changed.

SPOOL (SPON)

Syntax: SPOOL <filename>

Description: All output to the screen will also be sent to a file on disc with the specified name until the file is closed with the SPOOLOFF command.

SPOOLOFF (SPOFF)

Description: Cancels the echoing of all screen output to a file, having first closed the file.

SWAP (SW)

Description: Swaps between two documents in memory. All settings of the files and cursor, block markers etc are retained. See 'Two file editing' for full details.

TYPE (T)

Syntax: TYPE <filename>

Description: Used to 'type' the contents of a text file to the screen. The file is not loaded into memory, merely the contents displayed on the screen. This can provide a convenient means of viewing the contents of a file without loading it into memory. Whilst the file is being typed pressing STOP will pause the display. Pressing STOP a second time will cancel the command and any other key will resume.

(ii) Text manipulation

FIND (F)

Syntax: FIND <text> (<parameters>)

Description: The document will be searched for the first occurrence of the specified text, according to any parameters specified and the cursor positioned on the first character.
See chapter on Find and Replace for full details.

REPLACE (R)

Syntax: R <text> <newtext> (<parameters>)

Description: The document will be searched for the first occurrence of the specified text, according to any parameters specified, and the cursor positioned on the first character.

NUMBER (NUM)

Description: The purpose of this command is to add line numbers to, or remove line numbers from, the beginning of every line of text. This command will prompt for whether numbers are to be added or removed from the document. If the choice to add line numbers is selected, a starting line number and the value by which each subsequent number is to be incremented will be requested.
This provides a convenient method of writing BASIC programs, amongst other uses, using the facilities of the editor and finally adding the line numbers.

NUMBERB (NUMB)

Description: This command is similar to NUMBER, but only adds or removes numbers within the marked block.

TAB

Syntax: TAB < column(s) >

Description: Sets a tab stop at the specified column or columns. The last number in the list may be preceded by '*'. This causes tabs to be set at equal intervals up to column 128.

Example:

TAB 8, 15, *5 sets tabs at 8, 15, 20, 25, 30,.....

TAB without any parameters sets default tabs at every 8th column.

(iii) Printer control and Printing

The following commands determine the form that printing will take.

BM

Syntax: BM < number >

Description: Bottom Margin. Specifies the number of lines to be left blank at the bottom of each page.

INTERNAL (INT)

Description: Resets the printer output to the normal printer supplied with the PCW range.

PARALLEL (PAR)

Description: Selects the parallel (Centronics) printer port for the output of all printing.

PL

Syntax: PL <length>

Description: Sets the length of each page in lines.

PRINT (P)

Syntax: PRINT (num)

Description: This command prints the document in memory.

PRINTB (PB)

Description: Only the section of text defined by the block markers will be printed.

PRINTER (PR)

Syntax: PRINTER (name)

Description: Selects the printer driver to be used.

PRINTON (PRON)

Description: All output to the screen will also be echoed to the printer, after this command has been used, until PRINTOFF is used to cancel it. One particular use is to provide a printed copy of a disc catalogue.

PRINTOFF (PROFF)

Description: Cancels the echoing of screen output to the printer, which has previously been initiated by use of the PRINTON command.

SERIAL (SER)

Description: Redirects all printed output to the serial interface, for use with a serial interfaced printer.

(iv) Disc drive selection, cataloguing and disc formatting

A: (A)
B: (B)
C: (C)
D: (D)

Description: Select drive A, B, C, or D as the currently selected drive. Optionally, the colon may be omitted.

Note: Only valid if the specified drive is fitted and initialised.

M: (M)

Description: Select drive M as the currently selected drive. Optionally, the colon may be omitted.

Note: Only valid on the PCW computers.

DFORM

Description: Formats a disc to either CF2 or CF2DD format, depending on which drive is selected. On the PCW computer a disc in drive B will be formatted to CF2DD format and a disc in drive A to CF2 format. On the CPC6128 data format is always used.

Note: Both sides of a CF2DD disc are formatted at the same time, but when formatting a disc in A to CF2 format, it is necessary to format each side separately.

DFORMD

Description: This command will format a disc as CPC6128 Data format.

Note: PCW users should use this command if the disc will also be used on a CPC6128. On the CPC6128 this command has exactly the same effect as DFORM.

DRIVE (DR)

Syntax: DRIVE < drive letter >

Description: Selects the specified drive. This command will accept drives between A and P, and an error message will be given if the requested drive does not exist, or if it does exist but there is no disc in the drive.

Note: If any special drives are installed, such as a hard disc, which use a drive letter other than A, B, C, D, or M, then this command may be used to select the drive.

GROUP (GR)

Syntax: GROUP < number >

Description: Selects the specified group/user number as the one which will be used by CAT, LOAD, SAVE etc.

USER (U)

Syntax: U < number >

Description: Selects the specified group/user number as the one which will be used by CAT, LOAD, SAVE etc. An alternative to the GROUP command.

CAT (DIR)

Description: Performs a catalogue of the files on a disc. By default, with no parameters, it will catalogue all the files on the currently selected group of the currently selected drive.

Extensions: Filenames, drive letters and group numbers.

Syntax: CAT < ambiguous filename >
CAT < drive letter >
CAT < group/user number >

Description: Either another group OR another drive may be specified. Alternatively a filename may be specified using wildcards, optionally with a drive letter prefix.

Example: CAT B:*.LTR will catalogue all the files with a LTR suffix on drive B and group 0.
The files are listed in alphabetical order with the size of each file shown. The amount of free disc space is also shown. If this last figure becomes too small it will be often be necessary to erase backup files in order to save a file. The catalogue also displays a symbol by certain files:

Note: A '*' following the filename extension indicates a protected file (see PROTECT, below)

INFO

Syntax: INFO < ambiguous filename >

Description: The info command provides information about files. The filename used can include wildcards, so INFO *.SRC will give you information on all files with the extension .SRC. The information displayed is the file length, file type (i.e. document, program, system..) and its read or write state. RW means Read and Write. RO means Read Only.

(v) Disc file manipulation

COPY

Syntax: COPY <oldname> <newname>
COPY <ambiguous filename> (<group>) (<drive>)

Description: There are two variations of this command, the first of which will copy a file giving it a new name. The filenames may be prefixed with the drive letter to copy a file from one drive to another with a different name.

The second variation permits the use of 'wildcards', but the names cannot be changed. This allows the transfer of one or a number of files with some common feature, from the current group on any drive to any group on any drive. Either <group> or <drive> or both may be specified.

Examples: COPY B:OLDNAME NEWNAME
COPY *.TXT 1
Copies all files with suffix 'TXT' into group 1.
COPY B:*.* 2M
Copies all files on drive B (current group) to drive M group 2.

Note: Any existing file with the same name, in the destination drive/group, will be renamed with a '.BAK' suffix.

Note: Copying does not erase the original files, so if they are no longer required, ERASE must be used after the copying process.

DCOPY

Description: Calls an Arnor utility program which copies the contents of one disc onto another. This command will copy single sided single density (CF2) discs only. Requires DCOPY.COM to be present on one of the drives.

Note: The original contents of the disc will be erased.

Note: In order to copy CF2DD double sided discs, as used in drive B on the PCW8512, it is necessary to leave the editor and use the DISCKIT program which is on the System Utilities disc supplied with the computer.

ERACOPY (ECOPY)

Syntax: ECOPY <oldname> <newname>
ECOPY <ambiguous filename> (<group>) (<drive>)

Description: This is the same as COPY in every respect but one - if a file already exists with the same name as the file being copied, this file is erased before copying, whereas COPY renames this file as a backup file.

Example: ECOPY B:*.* A

ERASE (ERA)

Syntax: ERA <ambiguous filename>

Description: All files which meet the criteria of the filename will be erased. Wildcards are permitted and the drive letter may be specified as a prefix to the filename.

Note: This is a potentially destructive command and should be used only with care. One very useful version is to use ERASE *.BAK to erase all back up files from the disc in the current drive. ALT-f7 on the PCW (and CTRL-f9 on the CPC) will perform 'ERA *.BAK'.

RENAME (REN)

Syntax: RENAME <newname> <oldname>

Description: This command renames files on a disc. It does not move or change the file, merely renames it.

Note: If a file requires moving to another disc and renaming, the COPY command should be used and then the original file erased with ERASE.

(vi) File protection

ACCESS (ACC)

Syntax: ACCESS <ambiguous filename>

Description: Sets the status of a file or files to 'Read-write'. Wildcards are permitted. See PROTECT for details of the reverse operation.

PROTECT (PROT)

Syntax: PROTECT <ambiguous filename>

Description: Sets the status of a file or files to 'Read-only'. Wildcards are permitted. Files which have read only status can not be overwritten by subsequent files of the same name. An error message will be given if an attempt is made to do so. Protected files are indicated in the catalogue by an asterisk following the filename. See ACCESS for details of the reverse operation.

Note: PROTECT cannot stop files being erased if the disc is reformatted, or a complete disc is copied onto the disc, either with DISCKIT or the DCOPY command.

(vii) Phrase, Exec and Symbol commands

EXEC (X)

Syntax: EXEC <filename>

Description: This command causes the contents of the specified file to be treated as if they were being typed in at the keyboard (See Chapter on EXEC Files for full details).

LPHRASES (LP)

Description: Lists all the currently defined phrases between A and Z. See the chapter on 'Phrases' for full details.

PHRASE (KEY)

Syntax: KEY <letter> <phrase>

Description: Following the command should be a key letter (A-Z) which is the key that will be used with the EXTRA key, to recall the phrase. This should be followed by the phrase, which may consist of simple text or may be a combination of text, commands and Escape codes. See the chapter on 'Phrases' for full details.

SYMBOL (SYM)

Syntax: SYMBOL <char>,<n1,n2,n3,n4,n5,n6,n7,n8>

Description: The SYMBOL command allows you to redefine a character as it will appear on the screen. The first number following the command is the character to be redefined. The eight numbers following are the bytes making up the character. See your Basic manual for more information on SYMBOL as this command is identical.

(viii) Miscellany

CPM

Description: Quits straight to CP/M.

HELP (H)

Syntax: HELP
HELP <subheading>

Description: The command, 'HELP', used on its own will produce a list of the subheadings in which the commands are grouped. Entered with the appropriate subheading (subheadings may also be abbreviated to a single letter), it will list all commands relating to that subject, together with any abbreviations.

Note: The file ACOMMAND.HLP must be present to use HELP with a subheading.

PAUSE

Syntax: PAUSE

Description: This command is primarily intended for use in an EXEC file. See chapter on "EXEC files".

QUIT (Q)

Description: Quits the editor and returns to CP/M command mode. If a document is in memory and any changes have been made to it since it was loaded or last saved, a caution will be issued, warning that the document has not been saved and asking for confirmation of the desire to continue.

Note: Details of any files in memory, settings and phrases are retained on the temporary drive and if the editor is re-entered, the editor will be configured exactly as it was when exited.

(ix) External commands

External commands call other utility programs from disc. The program files specified must be available at the time the command is used. They may be on any disc drive - the editor will search all drives to find the file. The following utility programs are designed so that on completion of their task, a return is made to the editor with any text that was in memory at the time the command was called still intact.

CONFIG

Description: This command calls the the editor configuration utility program CONFIG.COM. This allows many of the default settings of the editor to be altered to suit the user. The file, CONFIG.COM must be available. See 'Utility programs' for full details.

DCOPY

Description: Calls a utility program which copies the contents of one disc onto another. This command will copy single sided single density (CF2) discs only. See 'Utility Programs' for full description.

Note: The original contents of the disc to which the files are being copied will be erased.

Note: In order to copy CF2DD double sided discs, as used in drive B on the PCW8512, it is necessary to leave the editor and use the DISCKIT program which is on the System Utilities disc supplied with the computer.

SETPRINT (SP)

Description: A utility program to create suitable printer drivers to enable any printer to be used with the editor. The file, SETPRINT.COM must be available. See 'Utility programs' for full details.

(x) Programming commands

AC

Description: Compiles links and runs a C program. If there are any compilation errors the program is not run.

Note: This command requires the Arnor C program files to be available.

ASM

Description: Assembles the file in memory. If there isn't one you are prompted for a file to assemble.

Note: This command requires the Maxam II Assembler program files to be available.

MA

Description: As ASM, but prompts for a filename unless one was specified following the command.

MON

Description: Runs which ever version of the Maxam II Monitor it finds first, passing the name of the file being edited as a parameter. If this file has previously been assembled the object code of that name is automatically loaded into the monitor.

Note: One or other of the Maxam II program files must be present on one of the drives.

MM

Description: As above but runs the large version of the monitor.

MSM

Description: As above but runs the small version of the monitor.

Note: The MM and MSM commands will not attempt to load in the object code from the program being edited.

Note: With the MM and MSM commands, the appropriate program file must be present on one of the drives.

RUNC

Description: Enters the Arnor C run time system.

Note: The Arnor C program files must be present on one of the drives.

Each of the above commands can be followed by a filename. The specified file will then be used by the called program. If "" is entered as the filename then the assembler and C will prompt you for a filename.

(xi) External programs

Other programs may also be called from within the editor. When this is feature is used, the text in memory will be saved to a temporary file and the name passed as a parameter to the program being called.
For example, typing '*BCPL' will call the compiler called BCPL (i.e. the program file BCPL.COM), which will then compile the source code which was in memory. Unless these programs have been written specifically for the purpose, they will not return to the editor automatically and this must be done by typing 'APED' from CP/M command mode. The temporary file will automatically be loaded back into the editor.

*

Description: Runs a specified CP/M Plus program.

Example: *BCPL

will run the BCPL compiler, passing the name of any file in memory as a parameter

c). Large Files

The editor is capable of handling large files very efficiently and the only limit on the size of the files which can be edited is the capacity of the disc drives. It must be remembered that under CP/M, large files cannot be totally loaded into memory at one time, and as editing continues and progress is made through a long document, the editor will automatically save parts of the document as temporary files.

As a result, it is preferable to start editing a large file with as empty a disc as possible. With the PCW computers, drive M is normally used as the drive on which these temporary files are stored. On the CPC6128, which does not have a memory drive, the temporary files are saved onto the text file disc. This would normally be drive B on a two drive system.

Note: it should also be remembered that there must be sufficient space on the text disc to save the file when editing is completed.

In the event that the file becomes so large that there is no room left for the temporary files to fit, a 'Disc full' message will be issued. If this happens, it will usually be possible to delete one or two files from the disc or drive to make room, before continuing. For example the disc might have copies of the help files on it, in which case deleting these would give more space. Once this situation has been reached, it is worth considering whether the file should be broken down into smaller parts if further editing is required.

Other than the points mentioned above, editing of large documents is exactly the same as editing any other document. It should also be remembered that the ALT-[and ALT-] commands move to the start and end of the text in memory, not the start and end of the whole document. With a small document this will be the same thing, but if the start or end of a long document is required, then ALT-[or ALT-] should be pressed a second time.

Important notes on large file editing.

1. It is important to ensure that a disc is present in the selected drive at all times and that it is not changed for another disc during the course of editing the document.

2. The editor saves temporary files with various names commencing with 'APED'. Under NO circumstances must any of these files be deleted. When the document is completed and saved, the editor will automatically delete the temporary files which are no longer required.

Are large files necessary?

Even though the editor can handle 'unlimited size' files, this is perhaps a suitable place to consider whether it might be more convenient and efficient to work with a number of smaller files. Rarely is there any NEED for a long document to be in one piece. For example: A long program can be broken down into a number of sections or subroutines.

Whilst it may appear that there are advantages to being able to work on one long file so that it can all be viewed and edited at the same time, there are a number of points which should be considered.

1. In the event of a catastrophe, such as a power failure, or accidentally deleting a file from a disc, if the text is in one long file, the complete file may be lost.

2. Due to the limited amount of memory available under CP/M, it is not possible to have the whole of a large file in memory at the same time and as progress is made forwards and backwards through the file, parts of it have to be saved to temporary files and other parts loaded. The editor has specially written routines which do this more efficiently than other programs, but it can still take a short time to jump from one part of a file to another, whereas with a smaller file this will to all intents and purposes be instantaneous.

3. It is usually easier to locate specific sections of text in a smaller file.

4. Usually only a relatively small part of a file will actually be worked on at a time and it is considerably quicker to load, and save smaller files.

d). Two File Editing

The editor provides the facility to work on two files at the same time. These files are maintained quite separately and are loaded and saved individually. Any operation can be carried out on one file without affecting the other, the cursor location and all markers being maintained for each file. Blocks of text can be copied between one file and the other.

This is an extremely powerful function and is controlled by only three commands, one of which is used from command mode and the other two from edit mode.

SWAP (SW)	: Command mode	-	Swap between two files in memory
ALT-O	: Edit mode	-	Copy block over from the other file
ALT-Y	: Edit mode	-	same function as SWAP

To load a second file, 'SW' should be entered from command mode and the current file will be switched, leaving an empty file. The second file should be loaded in the normal way. Switching between the two files will cause the information on the status lines to change to suit the current file, enabling easy recognition of which file is being worked on.

In edit mode, ALT-Y performs exactly the same purpose as 'SW', enabling quick switching between files.

The ALT-O (letter o) command is extremely useful, as it enables any part of the text of either file to be copied over to the other.

Before a block of text can be copied over, the block should be marked out using the markers in the normal fashion. Typing ALT-Y will swap files and the cursor should then be positioned where the text is required. If ALT-O is then pressed, the block will be copied across.

If the original text is no longer required, ALT-Y should be pressed again, to return to the original file, followed by CUT, to delete the original text.

Two file editing is also very convenient as a means of keeping notes, for later attention, during the course of editing a file. Press ALT-Y, make the note and ALT-Y again, to return to the original file.

Another use for ALT-O is for transferring text from one file to another - load the first file, type SWAP, load the second file and use ALT-O to copy the blocks required into the first file, before re-saving it. This is quicker than using SB (save block), loading the other file and merging the saved block of text into the file and finally resaving it.

e). SPECIAL CHARACTERS

The editor is capable of being used with most non-English languages and fully supports the use of accents and characters such as c-cedilla.

Characters containing accents may be typed in during the course of editing and will appear correctly on screen.

There are seven main accents which are required to cover the usual range of European languages and these may be obtained in the following way.

The base character should be entered first and then immediately followed by EXTRA and the number key which contains the required accent (the accents and their keys are listed below). The accent will then be positioned over the character. Accents may be used with any character, which permits the use of the editor with a number of languages which normally are not catered for. Welsh and many of the Eastern European languages are covered.

Note: CPC6128 users should note that CTRL-1 to CTRL-7 are used to obtain accents, instead of the EXTRA key and a number key. The special characters are obtained by pressing CTRL-0, followed by the appropriate letter key, or by pressing one of the function keys with either SHIFT or CONTROL (see below).

If an accent is required by itself, press space followed by the accent key. Should any of the accent characters be required frequently it is possible to re-define the keys to give just the accent.

Accents supported

PCW key	CPC key	Accent
EXTRA-2	CTRL-3	Umlaut
EXTRA-5	CTRL-5	Ring
EXTRA-6	CTRL-7	Acute accent
EXTRA-7	CTRL-6	Circumflex
EXTRA-8	CTRL-1	Grave accent
EXTRA-0	CTRL-4	Inverted circumflex
EXTRA-hyphen	CTRL-2	Tilde

Note: The keys used on the PCW are the same ones used under CP/M, with the exception that EXTRA-0, the 'Inverted circumflex', which is used by a number of Eastern European languages, is an additional accent.

In addition to these accents, which may be used on any character, a number of phrases are initially defined as special 'non-English' characters and in the case of the PCW, these are the same keys as are used in CP/M. A list of the keys to be

pressed is given below.

Note: The phrases may be redefined and care should be taken when selecting phrases, if any of these characters are required.

Summary of special characters available from the keyboard

The command LPHRASES displays all the characters available using the EXTRA key (or function keys on a CPC 6128).

PCW key	CPC key	Character	
f3	CTRL-f1	Lower case o slash	ø
f5	CTRL-f4	Lower case diphthong	æ
f7	CTRL-f7	Lower case c cedilla	ç
SHIFT-f3	SHIFT-f1	Upper case O slash	Ø
SHIFT-f5	SHIFT-f4	Upper case diphthong	Æ
SHIFT-f7	SHIFT-f7	Upper case C cedilla	Ç
EXTRA-A	SHIFT-f5	Superscript a	ª
EXTRA-C	CTRL-f0	Copyright	©
EXTRA-O	SHIFT-f6	Superscript o	º
EXTRA-P	CTRL-f8	Paragraph symbol	¶
EXTRA-S	SHIFT-f0	Eszett	ß
EXTRA-Y	SHIFT-f3	Yen sign	¥
EXTRA-?	CTRL-f5	inverted ?	¿
EXTRA-!	CTRL-f6	inverted !	¡
EXTRA-<	SHIFT-f8	French open quotes	«
EXTRA->	SHIFT-f9	French close quotes	»
SHIFT-ALT-←	CTRL-V ←	Left arrow	←
SHIFT-ALT-→	CTRL-V →	Right arrow	→
SHIFT-ALT-↑	CTRL-V ↑	Up arrow	↑
SHIFT-ALT-↓	CTRL-V ↓	Down arrow	↓

f). Phrases and function keys

Phrases are pieces of text which can be stored and used at any time with a single key press. The keys used to recall phrases are the keys marked 'A to Z' on the main keyboard when used in conjunction with the EXTRA key. Function keys are essentially the same, but use the special function keys on their own and in conjunction with the ALT, SHIFT and EXTRA keys.

There are 31 expansion tokens which by default are allocated to the keys EXTRA-A to EXTRA-Z and a number of other keys. Several of these tokens are also allocated to the function keys, duplicating a number of the letters. Some of these tokens are already defined and cannot be changed, leaving 26 tokens which may be defined by the user. By default, many of these tokens are pre-defined to give a variety of European characters, such as 'C, cedilla' and 'AE diphthong', but may be redefined by the user if not required for that purpose.

Each of these tokens can be allocated a string of text or codes up to 200 characters long.

(i) Predefined tokens

The following tokens are predefined by the editor and may not be changed. Each of these selects command mode and executes a command.

PCW	6128	Definition
f1	CTRL-f3	CAT
ALT-f7	CTRL-f9	ERA *.BAK
FIND	CTRL-f2	FIND
EXCH	SHIFT-f2	REPLACE
EXTRA-ENTER	CTRL-ENTER	EXEC EXFILE

(ii) Phrases and function key definitions

As far as the editor is concerned, there is no difference between phrases and function key definitions. They are both merely strings (of text or codes) and any difference would be in the use to which they were put, rather than their format. For example, function keys would probably be used to carry out tasks or functions, whereas phrases would be used to store text to be incorporated into documents, though there is no reason why they should not be used for other purposes.

A string has a maximum length of 255 characters, subject to the total buffer size and the free space remaining in it. It may contain any characters and control codes. Any normal text may be typed in from the keyboard as usual, but in order to be able to enter control codes, an escape code must be used to inform the editor that the characters which follow constitute a control code. The escape code used by the editor is the upwards pointing arrow (↑). This is obtained by pressing EXTRA-: (colon) on the PCW. It is used to allow entry of the following:

↑ <number> ↑	Inserts the code specified by the number. The code may be entered in decimal, eg. ↑13↑, or hexadecimal, in which case it must be prefixed by either # or &, eg. ↑&0D↑.
↑ <letter>	is translated as a control code between 1 and 26 eg. ↑A would be translated as Ascii code 1, ↑B as 2, etc.
↑↑	is translated by PROTEXT as a single up arrow. This must be used if an arrow is required in the string.

Note: When specifying a code as a number, it must be both prefixed and suffixed with the escape code character (↑), but in other cases, it is only necessary to prefix the character with the escape code. This is because a number could consist of from one to three characters.

As an example of how one would use a control code, if a key was to be defined so that when it was pressed it automatically did a catalogue of drive A followed by a catalogue of drive B, the following string would be used:

CAT A↑13↑CAT B↑13↑

'13' is the code for a carriage return, which would normally be given when the RETURN key is pressed. As CAT requires the RETURN key to be pressed, the codes are inserted into the string. Alternatively ↑M could be used instead of ↑13↑.

Details of the most useful codes are given in an appendix at the end of the manual, but in the unlikely event that a full key translation list is required, this is available from Arnor on request.

(iii) Phrase commands

There are two commands which are directly connected with phrases and are used from command mode:

PHRASE (KEY)

Syntax: PHRASE <letter> <string>
KEY <letter> <string>

Description: PHRASE and KEY are alternative names for the same command. This command allows temporary strings to be created at any time. The command is used from command mode and the letter must be a letter between A and Z, followed by the string of text or codes, which should be wrapped in quotation marks.

Note: If it is required to cancel a key definition, this can be done by using a null string ("") following the key letter in the parameters. This may prove useful when a number of phrases have been defined and the buffer is too full to take any further definitions. Any phrases which are no longer required can be discarded in this way.

LPHRASES (LP)

Description: Use of this command will list the contents of all the defined phrases between A and Z. Where a phrase contains a code between 0 and 31 or between 192 and 255 this will be displayed in escape code form, e.g. ↑9↑.

(iv) Storing phrases for regular use

The command PHRASE, which enables temporary phrases to be defined has already been described and is very useful for quickly defining a phrase during the course of editing a document, but once the program is left, these phrases will be lost and would require re-entering the next time that the editor was used.

The editor has another method of defining phrases and function key definitions, which enables users to keep one or more files of definitions on disc and to load them as and when required. This is done through the use of an EXEC file.

A set of phrases should be saved with an appropriate name. Only those keys required and their definitions need to be in the phrase file and any existing phrases will not be changed or deleted unless redefined by the new ones. When they are required, it is only necessary to go into command mode and use the following command:

EXEC <filename>

where <filename> is the name of the file containing the phrases. This will automatically allocate them to the specified keys.

(v) Using phrases and function keys

Once a phrase or function key has been defined by either of the above processes it may be used within the monitor or editor at any time by pressing the appropriate key. Any of them may be used either when in edit mode, or command mode. The most convenient arrangement would probably be to use the function keys for commands which would be used in command mode and the keys 'A' to 'Z' for strings of text to be used in documents.

Phrases are called by pressing EXTRA and one of the letter keys between 'A' and 'Z', which gives 26 different possibilities. The function keys may be used either on their own, or in conjunction with SHIFT, ALT, EXTRA and SHIFT-ALT, which gives 20 possible combinations on the PCW.

When a phrase key or function key is pressed, the contents of the string will be entered into the document (if in edit mode), or the command line (if in command mode), as if it had been typed in at the keyboard, and any control codes will be acted upon.

On the CPC6128 the 10 function keys f0 to f9 may be used either with SHIFT or with CONTROL. The phrases are obtained in edit mode by typing CTRL-0 (zero) followed by a letter. Only the function keys may be used in command mode, but these are set up to provide the various European symbols.

Note: Most of the function keys are allocated the same tokens as the keys A to Z and redefining one will also change its equivalent. It is not possible to have different contents in each of them. To set up a function key the KEY command is used with the corresponding letter. A table listing the function keys and the corresponding letters is given as an appendix.

g). Exec files

(i) What is an EXEC file?

An EXEC file is a file which may contain text, commands and codes and which, when called with the EXEC command, will be read by PROTEXT and the contents treated and acted on, as if they had been entered at the keyboard.

They are created in just the same way as any other text file, but what makes them different is the content of the file and the way it is used later.

(ii) Creating an EXEC file

Creating an EXEC file is extremely simple and is done by just typing the required text in, as would be done with any document, but there are a number of special features which permit codes to be inserted into the text, which the editor will understand to be instructions to do certain things.

In addition to ordinary text, any of the editor's command mode commands may be used as well as any valid code between 0 and 255.

Codes must be entered in a special way, otherwise the editor will consider them to be ordinary text. A special 'Escape character' is used to tell the editor that the following character(s) is/are a code and the escape character used is the vertical bar (¦). This is obtained by pressing EXTRA-. (full stop) on the PCW. The escape codes are exactly the same as used in phrase definitions except that the bar (¦) is used instead of the arrow (↑).

The easiest way to describe the sort of uses to which an EXEC file might be put, is to give one or two examples. The examples given are intended to show the sort of things that can be done, rather than be particularly useful:

Example to change every occurrence of a certain word to another word in a number of files.

```
L file1
R "buffer" "BUFFER" GA
S¦13¦
L file2
R "buffer" "BUFFER" GA
S¦13¦
L file3
R "buffer" "BUFFER" GA
S¦13¦
```

In the above example, 'L file1' is the command to load a file called 'file1' and 'R "buffer" "BUFFER" GA' is the REPLACE command used with the options to make it global and automatic. 'S ¦13¦' is the SAVE command. The process is then repeated for 'file2' and so on.

Note: When a new line is used, the editor will take this to mean that a carriage return character (CR) is required, as would normally be given by pressing RETURN after entering the command. In the case of the lines concerned with saving, escape characters have been used to insert an extra CR code into the file. The reason for this is that if a file is to be saved with the same name, then RETURN is pressed once after entering the 'S' and again to confirm the same filename.

When complete, the file should be saved with an appropriate name. Entering 'EXEC <filename>', from command mode will automatically execute the file of that name whenever required.

(iii) Creating a phrase file

This is easily done by using the PHRASE (KEY) command in an EXEC file. This is an example of a phrase file, to be used to define phrases and function keys in the editor:

```
KEY C ""
KEY G ""
KEY B "This is a remark which can be inserted into the text"
KEY D "The EXTRA key and the appropriate letter should be pressed"
KEY A "CAT A ↑13↑CAT B↑13↑"
```

In the above brief example, keys B and D are straightforward examples of text to be inserted when the appropriate key is pressed.

Keys G and C are defined as null strings. This will have the effect of removing any existing definition from keys C and G. This may be desirable if a second phrase file is being loaded, when a number of keys are already defined, otherwise the phrase buffer may become full before all the new definitions are loaded.

Key A is an example of the sort of definition which would be used for a function key and in this case would perform a catalogue of drive A, followed by drive B when SHIFT-f1/2 was pressed.

It should also be noted that '↑' has been used, rather than the '¦' symbol, as these commands are simulating entry of the phrases at the keyboard and phrases require the '↑' escape character.

Note: Because the EXEC file is executing a command to define a string, it is necessary to specify the CR at the end of the command if one is required when the function key is pressed, as the CR which is implied at the end of each line of an EXEC file will serve only to execute the KEY command.

It is recommended that phrase files should be saved with a suitable suffix to identify them, say '.PHR'.

(iv) Commands related to EXEC files

EXEC (X) Execute a file

Syntax: X <filename>

Description: The file specified will be opened for reading and the contents read will be treated as if they were input from the keyboard, until the end of the file is reached, at which time normal operation will continue.

PAUSE Cause the editor to go into a 'waiting' condition.

Syntax: PAUSE

Description: When this command is read by an EXEC file, the program will halt until a key is pressed. Optionally a message will be displayed. This is useful during the course of an EXEC file being executed, as it will permit discs to be changed and messages to be displayed before continuing execution.

(v) Using EXEC files

EXEC files may be used at any time by typing EXEC from command mode, followed by the name of the file to be executed. If a file called 'EXFILE' is present on the currently selected drive, it may be executed at any time by pressing EXTRA-ENTER (CTRL-ENTER on the CPC6128).

There is one further feature which can be very useful. If the 'less than' symbol is used to prefix an EXEC filename when the editor or monitor is called, it will be taken to mean that all input should be taken from the specified file, until the end of the file is reached. For example, 'APED textfile <exfile' would load a text file into memory and then execute the EXEC file, which if required could then carry out operations on the text file, such as replacing text.

6. CONFIGURATION UTILITIES

Some features of the editor may be re-configured to suit individual requirements and these may be changed by using the CONFIG.COM utility program. In addition, the program SETPRINT.COM is also provided and may be used to modify existing printer drivers, or to create new drivers to suit printers which use codes different from those on the supplied drivers.

These two utilities were originally written for PROTEXT. To retain compatibility between the editor (APED) and PROTEXT, the full programs, as supplied with PROTEXT, are included on the disc. This means that many of the functions are not relevant to the program editor.

It is recommended that you get used to using the editor before attempting to change its configuration. If you create a configuration you don't like you can either delete PROTEXT.CFG or re-enter CONFIG to adapt it.

In addition, a third utility program, DCOPY.COM is supplied and this may be used to copy the contents of any CF2 **single** density disc onto another disc for the purpose of making a back up copy.

Note: CPC6128 users with only one disc drive must save any files to disc before using any of the utility programs and on completion it will be necessary to load the file back into memory in the normal way. If the file is not saved before using the utility, it is probable that part or all of the document will be lost.

All three programs may be used either from within the editor, or directly from CP/M command mode by typing their name. The appropriate program must be available on one of the drives at the time the utility is called from the editor, or the current drive from CP/M, otherwise an error message will be given.

a). DCOPY

It is good practice to keep back ups of any important files as it is always possible for accidents to happen, or discs to fail. It is also good practice to make backups regularly and systematically. DCOPY provides a complete copy of the original disc, including any .BAK files.

Warning: DCOPY will copy the entire contents of one disc onto another. If a blank formatted disc is accidentally copied onto a disc which contains files, the result will be two blank discs, so care should be taken to ensure that the 'write protect' tab on the original disc is set in the open position to ensure that this cannot occur.

Note: The COPY option of DISCKIT should be used on the PCW to make back ups of any CF2DD format discs of the type used by the second drive (Drive B) on the PCW8512 and expanded PCW8256. The DCOPY program is only intended to be used to copy single sided CF2 discs.

Once the program is called, instructions are given about which drive should contain the original disc and also which drive to put the disc to be copied into. Users of machines with only one drive will be given instructions about which disc to insert at the appropriate times.

Before any copying is undertaken, the write protect tab should be opened on the disc to be copied, to make sure that it cannot be accidentally overwritten. When ready, press 'S' to start, or press 'C' to cancel. Copying will then take place automatically. On completion, the opportunity to copy another disc is offered. Selecting 'N' at this point will return to command mode.

Note: This program takes an exact copy of the entire contents of a disc. Any files on the disc onto which the original is to be copied will be erased, so it is important to ensure that there is nothing of importance on the backup disc before commencing copying. If it is required to copy only a number of files onto another disc which already contains files, the the editor commands, COPY or ERACOPY, should be used to copy the relevant files across.

b). CONFIG

CONFIG allows certain of the features of the editor to be configured to suit individual preferences if the default settings are not considered suitable.

The most likely options to require alteration are:-

Default drive for text
Default group for text
Temporary text drive
Insert/overwrite mode
Set continuous/single sheet printing
Undelete buffer size
Set printer driver options
Set name for Autoexec file

Only some of the many options available in CONFIG are relevant to use with the editor and these are described below. Changes made to options which are not relevant will be ignored by the editor.

Option 2 from the main menu is NOT relevant to the editor.

The required main menu option should be selected by pressing the appropriate number key and the screen will clear and be replaced by a further set of options.

(i) Editing the options

Once the selection has been made from the main menu with the number key, all the options within the new menu are selected by one of two methods, depending on the selection made. The 'Set keys' options are slightly more complex and are described separately.

Details of the current values of each option are shown and in the event that there is no existing PROTEXT.CFG file, then the values shown will be the default ones provided by the editor. The up and down cursor keys are used to select which of the options are to be changed

The STOP key (ESC on the CPC6128) can be used, at any time, to go back to a previous menu. If a beep is heard, then this means that an illegal value has been entered, or else the key that has been pressed is not relevant. In much the same way that an illegal value may not be entered, it is not possible to move from an entry until a legal value has been provided.

Some options have an easily defined set of possible values (such as yes/no, or internal/serial/parallel printer), in which case using the right cursor key will move forwards through the options and using the left cursor key will move back through them. Once the required value has been selected, the up and down cursor keys may be used to move on to another parameter.

There are also a number of options which have no set range of parameters, such as those requiring the name of a file. In this case the parameter to be altered is selected, as before, with the up and down cursor keys and then the value required is typed in.

(ii) The Set keys options

The two options to 'Set keys' are slightly different and require a key number entering, to specify which key is to be edited. Each key on the keyboard has an individual 'key number' and these may be found from the diagram in the computer's User Manual. Once the key number has been entered and RETURN pressed, a table will appear.

This table is split into three parts vertically, with the left columns showing the key number, the middle showing the key values as ASCII characters and the right side as hexadecimal codes.

The PCW 8256/8512 key table has five values for each key, representing that key when pressed on its own, with EXTRA, with SHIFT, with ALT, and with ALT and SHIFT simultaneously.

The CPC6128 table has three values - the key on its own, with SHIFT, and with CONTROL.

The cursor will be positioned on the ASCII character of the first column for the selected key. The left and right cursor keys may be used to move backwards and forwards across the ASCII characters. Pressing the required key on the keyboard will change the setting in the table to the new value. In addition, it is possible to move up and down through the key numbers by using the up and down cursor keys.

Normally this is the easiest way to change which characters are allocated to a key, but there are a considerable number of characters and codes which cannot be keyed in directly from the keyboard, as they do not have their own key. Pressing the TAB key will switch the cursor into the 'Hex code' side of the table and the hexadecimal number representing the character may then be typed in. The computer's User Manuals give full details of the hex values of characters and an Appendix at the back of this manual lists the codes for phrases and function keys, as well as the codes for the command keys.

When editing is complete, the STOP key may be pressed to back out to the previous menu.

The remainder of this chapter gives details of the various options available from the main menu and their functions.

(iii) Set editing options

This option enables the user to configure a number of features for use within edit mode, so that when the editor is first loaded the specified features will be in operation. Only the relevant configurable options are listed below, with brief notes describing those options for which the use may not be self evident. The options should be selected and altered by the methods described above.

Default drive for text
Default group for text
Insert/overwrite mode
Help lines on
Tabs and returns displayed
Spaces displayed
Undelete buffer size (1)
Cursor flash rate (2)
Temporary text drive (3)

1. Undelete buffer size - This specifies the amount of memory which must be set aside to store deleted text. The larger the buffer, the less memory will be available for the text.
2. Cursor flash rate - The larger the number, the slower the flash rate.
3. Temporary text drive - This option defines the disc drive on which the editor will store those parts of a text file which are not in memory.

Set printing options

The only relevant option is 'Continuous printing on', and this determines the sequence of codes to be sent to the printer at the start of printing. If the printer driver is 'simple' then nothing will be sent, otherwise the codes in the printer driver for continuous or single sheet will be printed.

(iv) Set general options

These options are concerned with setting the key repeat rates and the screen colours (normal or inverse on the PCW). The expansion string buffer size governs the size of the buffer which holds phrases as well as any function key definitions. Note that the bigger the buffer, the less memory will be available for text.

Expansion token buffer size
Key repeat rate
Key startup delay
PCW 8256/8512 colour
CPC 6128 border colour
CPC 6128 foreground colour
CPC 6128 background colour

(v) Set keys for PCW8256/8512

This option will rarely be needed, particularly by English speaking users, and is primarily included for the benefit of anyone who might prefer to re-configure certain keys on the keyboard to perform different functions. The most likely need for this would be to arrange for special symbols or foreign characters to be located on more suitable keys.

(vi) Set keys for CPC6128

This option serves the same purpose as c), and is for use with the CPC6128.

(vii) Set printer driver options

This option allows two default printer drivers to be specified for the editor to load (one for PCW8256/8512, one for CPC6128) automatically. The maximum amount of memory to allocate for control codes and for character translations may also be specified (See SETPRINT). Note that if more space is reserved than necessary, it will reduce the amount of memory available for text.

(viii) Set name for AUTOEXEC file

When the editor is first loaded, it will look for an EXEC file to execute. By default, this is a file called EXFILE. This option permits another file with a different name to be specified instead of EXFILE.

(ix) Save configuration

Once all the required options have been selected and values specified, the configuration details can be saved to a file by using this option. The configuration file will be saved with the name PROTEXT.CFG.

Note: The CFG file is saved with the name 'PROTEXT.CFG' and is exactly the same as a CFG file created with PROTEXT. If a suitable CFG file, created with PROTEXT is available, it may be used instead of configuring one specially for the editor.

Note: Only one configuration file may be present on a disc, as it must have the name PROTEXT.CFG if it is to be automatically configured on entry to the editor.

Note: Once a configuration file (PROTEXT.CFG) has been created, it should be copied onto the other side of the Start of Day disc, so that it is available when the editor is loaded.

(x) Quit configuration program

Using this option will return control to the editor command mode (or CP/M if CONFIG was loaded from CP/M). If any changes have been made to the configuration, but the Save configuration option has not been used, a request will be made for confirmation that the changes that have been made are to be discarded.

c). SETPRINT

A 'printer driver' contains information for the editor about the printer being used and the way it is to be used. This information is contained in a file with the extension '.PTR'. the editor comes with two printer drivers already on the disc.

PCW.PTR - specifies character translations for the PCW internal printer.
EPSON.PTR - specifies character translations for a standard Epson compatible printer.

The editor also includes the details of a 'simple printer', i.e. a printer which recognises no character translations.

Unless CONFIG has been used to specify a different default printer driver, the editor, when used on a PCW8256/8512, will use PCW.PTR to print to the internal printer, or if it fails to find it, will treat the internal printer as a 'simple printer'. Similarly, if no default printer driver is specified for the CPC6128, the editor will assume a 'simple' parallel printer.

If an Epson compatible printer is to be used, then CONFIG should be used to specify 'EPSON.PTR' as the default printer driver.
If the printer being used is not one covered by the supplied printer drivers, it is recommended that SETPRINT is used as soon as an understanding of the requirements has been gained and after studying the printer control code section of the printer manual, in particular.

Once the program has loaded, the screen will clear and an opening menu will appear, listing the available options. The only options of any interest for the editor are:-

1 - Set printer options
2 - Set serial options
4 - Set character translations
6 - Load printer driver
7 - Save printer driver
0 - Quit SETPRINT

The option required should be selected by pressing the appropriate number key and the screen will clear and be replaced by a further set of options. The up and down cursor keys are then used to select which of these options are to be changed.

If SETPRINT is entered from the editor, then the printer driver currently loaded printer driver will automatically be loaded into SETPRINT and the current values of each option will be displayed. If there is no printer driver loaded, or SETPRINT is entered from CP/M, it will be necessary to use option 6 to load the required printer driver.

(i) Editing the options

The STOP key (ESC on the CPC6128) can be used at any time, to go back to a previous menu. If a beep is heard, then this means that an illegal value has been entered, or else the key that has been pressed is not relevant. In much the same way that an illegal value may not be entered, it is not possible to move from an entry until a legal value has been provided.

Some options have an easily defined set of possible values (such as yes/no, or internal/serial/parallel printer), in which case using the right cursor key will move forwards through the options and using the left cursor key will move back through them. Once the required value has been selected, the up and down cursor keys may be used to move on to another parameter.

There are also a number of options which have no set range of parameters, such as those requiring the name of a file. In this case the parameter to be altered is selected, as before, with the up and down cursor keys and then the value required is typed in.

Some options require a series of codes to be entered and these can be entered either as text or as numbers. The top of the screen will display either the message, 'text' or 'numeric'. In text mode typing any normal character will insert that character as a code to be sent; some characters cannot be entered in this way, such as ESC (decimal 27). The TAB key is used to toggle between text and numeric input, so in order to input codes of this sort, numeric mode should be toggled on.

Once in numeric mode there are also two options, decimal and hexadecimal (the top of the screen will display 'decimal' or 'hex'). ALT-TAB will switch between decimal mode and hexadecimal mode. When entering numbers, they should be separated either by commas or spaces.

In all modes, the cursor keys will move to different parts of the code in the normal way. ALT with left and right cursor keys will move to the beginning and end of the line of codes; ALT-E will delete from the cursor position to the end of the line.

Editing 'Character translations'

In the case of the character translations, a table will appear showing which characters have already been redefined to other sequences of codes. The cursor keys should be used to move through the table and the codes assigned to the particular character the cursor is on will be displayed in the lower part of the screen.

The character translation table can only display half of the characters at a time and the other half may be selected or reselected, at any time, by pressing the TAB key.

Pressing RETURN when the cursor is positioned on the required character will select it, and the cursor will move down to the displayed codes to enable them to be edited. Editing and entry of the codes is carried out as described above. Pressing STOP will complete entry of the codes and return to the main table.

When editing is complete, pressing STOP a further time will return to the main menu.

(ii) Set printer options

These are general options and are used by the editor to determine in what format and where the text is to be sent when printing takes place. It is important when creating a printer driver, or modifying an existing one, that as many of the relevant questions as possible should be answered. It may not be possible to answer all of them, as some of the features may not be available on the printer being used.

The options relevant to the editor are:-

Printer type
Continuous paper code
Single sheet code

By default, if no printer driver is loaded, the only setting made by the editor will be 'unknown' printer type. No translations are defined.

Printer type

The setting of this option determines which of the printer ports will be used when the printer driver is loaded and the setting selected should be the relevant one for the printer to which the printer driver refers. The options available are 'parallel', 'serial', 'internal' and 'unknown'. The 'internal' option refers to the standard PCW printer and is not relevant for the CPC6128. If 'unknown' is selected, then no change will be made to the current setting when the printer driver is loaded.

Note: Only the 'internal' setting will be relevant on the PCW8256/8512 unless a serial/parallel interface has been fitted to the computer. On the CPC6128, the only relevant setting will be 'parallel' unless a serial interface has been added to the computer.

Continuous paper code
Single sheet code

Continuous paper and single sheet codes are supported by some printers and one has the effect of turning the other off. The main use of these codes is to set the correct mode of operation for the PCW printer and is unlikely to have any relevance for other printers.

(iii) Set serial printer options

This option only requires setting if the printer being used has a serial interface and is used to set the values of the following parameters:-

Baud rate	9600
Number of data bits	8
Parity	NONE
Number of stop bits	1
Mode Selector	0

The above options are all self explanatory, except for the last one, Mode selector'. This option is used to set whether handshaking is 'on' or 'off'. The default setting is 0 (zero), which is 'off' and the setting for 'on' is 255. If the serial printer being used requires handshaking, then the value should be changed to 255.

The default values, shown in brackets above, will suit most serial printers, but, if necessary, they may be changed to suit those available on the printer. It is important that the settings for the editor and those for the printer are the same. These options are all of the 'limited set' type and the left and right cursor keys should be used to rotate through the options.

(iv) Set character translations

This can be used to redefine one character either as another character or as a sequence of codes. This is useful for printing characters that do not have a standard ASCII code, or those that a printer does not normally support. The printer manual should be studied for details of how to create new symbols.

Users of daisy wheel printers may well find this option to be particularly useful, as a number of them have the less frequently used characters allocated values different from the more common standards.

Note: The PCW.PTR printer driver already has a number of characters translated, or redefined and all characters between ASCII 32 and 127 and also those characters listed in the PCW user manual, between 160 to 254, may be printed without any alterations being required. If the PCW.PTR driver is modified slightly to adjust the few codes which differ from the 'Epson' standard, it will be possible to use this printer driver with an Epson compatible printer, to print all of these characters as well.

Note: The code 255 decimal, the 'equivalent' sign should not be used or allocated to any character.

Any characters which have been redefined will be displayed in the table in inverse video.

A common problem is printing the '£' sign. If the printer is not printing this (it often comes out as a '#'), the character number for '£' should be ascertained from the printer manual and the '£' character redefined to be printed as this character number.

Note: If the CPC6128 is being used and the character number is greater than 127 (hexadecimal &7F) it will not be possible to print the character by this method, as the CPC6128 only has a '7 bit' printer port, which means that only codes lower than 128 may be sent to the printer. There are two possible methods of overcoming this limitation. Some printers have a special code, sometimes called 'Set MSB' or 'Set MSB to 1' and if so, these codes should be sent, then the code for the character and finally the codes to 'reset top bit' (or 'Reset MSB' or 'Set MSB to 0'). Alternatively, if the character is one that is contained in the printer's foreign character sets, it is possible to send the codes required to change the 'nationality', followed by the character and finally the codes to return the 'nationality' to its original setting.

(v) Load printer driver

Existing printer drivers may be loaded for modifications by use of this option. The default filename extension for printer drivers is '.PTR'.

(vi) Save printer driver

This option permits the modified or newly created printer driver to be saved for future use. The name by which the printer driver is to be saved, should be entered. The default filename extension for a printer driver is '.PTR' and it is recommended that this is used.

Note: Printer drivers should be saved onto both sides of the 'Start of Day' disc, so that they are available when the editor is loaded and also when SETPRINT is loaded.

(vii) Quit SETPRINT

If any changes have been made to the driver and the save option has not been used, SETPRINT will request confirmation that any changes are to be discarded before returning to the editor command mode or CP/M.

APPENDICES

A1. SUMMARY OF COMPILER COMMANDS

Description of abbreviations

afn	ambiguous filename (including wildcards)
cmd	runtime commands
d	drive letter
fn	a filename
lf=	optional link filename for resulting file
lfl	list of link files to be used
nfn	new filename
ofn	old filename
of=	optional object filename for resulting file
opts	optional parameters
sfn	source file

Parameters in angle brackets '<..>' are mandatory, whilst those in normal brackets '(..)' are optional.

a).Redirections available with the compiler

stdin, *stdout* and *stderr* may be redirected in the following ways:

<filename	redirects stdin to read from a file
>filename	redirects stdout to a file
>>filename	redirects stdout, appending to an existing file
#filename	redirects stderr to a file
#>filename	redirects stderr, appending to an existing file

b). Commands from the editor or CP/M

AC	compile the current text in memory
AC <fn>	compile a named file
RUNC	enter runtime system interactively
RUNC <cmd>	use runtime commands. See 'Run time system'
RUNC <fn> (opts)	run an already compiled program

c). Run time system

(i) Built in commands:

A	select drive A
B	select drive B
M	select drive M
Q	quit the runtime system

(ii) Run time command programs

	COMPILE <fn> (opt)	the source code compiler
	LINK (of=) <lfl>	the linker
	JOIN (lf=) <lfl>	join two or more link files into one link file
*	DIR (d)	utility to list a disc directory
*	DUMP <fn> (ofs)	hex/ASCII dump of file (opt. offset within file)
*	ERA <afn> (V)	erase specified files on disc (opt. Verify)
*	REN <nfn> <ofn>	rename specified file
*	TYPE <fn> (ofs)	type file to screen (opt. offset within file)

Note: Commands marked with an asterisk '*' are similar to the commands with the same name in the editor.

d). Compiler

Syntax:

COMPILE <sfl> (opts)

Options:

-d defines a macro
-g suppresses creation of the global table.
-l automatically link after a successful compilation
-m sets maximum errors reported before compilation abandoned
-q suppresses sign on message and summary information
-t sets drive for temporary files.
-w suppresses compiler warning messages.

e). Linker

Syntax:

LINK (of =) <lfl> (opts)

Options:

-l	suppress linking of standard library (STDLIB.L)
-n	list functions whilst linking, in form:
-q	suppresses sign on message and summary information
-r	automatically run the program after linking

f). Joiner

Syntax:

JOIN (lf =) <lfl> (opts)

Options:

-n	list functions whilst joining, in form:
-q	suppresses sign on message and summary information

APPEN 1-4

2. LIBRARY FUNCTIONS

The library functions are collected into groups in this section, according to their purpose. Each function is accompanied by a brief description. The degree of 'portability' is indicated by the following abbreviations:

S	Standard, as defined by Kernighan and Ritchie, or found in virtually all C systems.
C	Commonly found in many C systems.
A	Arnor C specific.

Three 'standard' library files are provided with Arnor C. The difference between them being the number of functions included. The 'mini' library contains a minimum number of functions. The 'small' library contains those functions indicated, as well as all the functions in the 'mini' library.

A separate library of mathematical functions is also provided.

The second abbreviation indicates the library in which the function is defined:

Min	Defined in all 3 standard libraries
Small	Defined in *MINLIB.L* and *SMLIB.L*
Maths	Defined in *MATHS.L*
Macro	Defined as a macro in *STDIO.H*

The full library contains all the functions listed, except those indicated as being in the maths library.

The next chapter gives full details of every function and is listed in alphabetical order for easy reference.

By convention, those functions with names that start with the underline character are primarily for the use of other functions, and would not normally be needed in a C program.

High level I/O functions

clearerr	S	Macro	clear file error flag
fbinary	A		set file to binary mode
fclose	S	Small	close file using file pointer
feof	S	Small	check for end of file
ferror	S	Macro	check file error flag
fflush	S		flush buffer
fgetc	S	Small	input character
fgets	S	Small	input until end of line
fileno	S	Macro	convert file pointer to file handle
fopen	S	Small	open file using file pointer
fprintf	S	Small	formatted output
fputc	S	Small	output character
fputs	S	Small	output string
fread	S	Small	input n characters
freopen	S		open file using existing file pointer
fscanf	S		formatted input
fseek	S	Small	seek to location in file
ftell	S	Small	get current location in file
ftext	A		set file to text mode
fwrite	S	Small	output n characters
getc	S	Small	input character
getchar	S	Macro	input character, using stdin
gets	S	Small	input until end of line, using stdin
getw	S	Small	input word
printf	S	Small	formatted output, using stdout
putc	S	Small	output character
putchar	S	Macro	output character, using stdout
puts	S	Small	output string
putw	S	Small	output word
rewind	S		seek to start of file
scanf	S		formatted input, using stdin
sprintf	S		formatted output to a buffer
sscanf	S		formatted input from buffer
ungetc	S	Small	put back last character read

Low level I/O functions

file I/O			
close	S	Min	close file
creat	S	Macro	create file
lseek	S	Min	seek to location
open	S	Min	open file
read	S	Min	input n characters
write	S	Min	output n characters
_fchret	A	Min	return character
_feof	A	Min	test for end of file
_finch	A	Min	input character
_foutch	A	Min	output character
_ftell	A	Min	get current location
screen output			
cursoff	A	Min	turn off cursor
curson	A	Min	turn on cursor
getcurs	A	Min	find location of cursor
getwin	A	Min	find location of window
invoff	A	Min	turn off inverse video screen display
invon	A	Min	turn on inverse video screen display
putch	C	Min	output a character to screen
rdmatrix	A	Min	read character matrix
selwin	A	Min	select window for output
setcurs	A	Min	position cursor on screen
setwin	A	Min	define a window
unwrchar	A	Min	read character from screen
wrchar	A	Min	output any character to screen
wrmatrix	A	Min	define character matrix
keyboard input			
escoff	A	Min	disable escape checking
escon	A	Min	enable escape checking
getch	C	Min	input a character from keyboard
getche	C	Min	input character and echo to screen
kbhit	C	Min	check for key pressed
ungetch	C	Min	return character to keyboard
printer output			
busypr	A		check whether printer is busy
prch	A		print character
setpr	A		select printer output

String functions

index	S	Small	locate character from start of string
strcat	S	Small	concatenate two strings
strchr	C	Small	same as index
strcmp	S	Small	compare two strings
strcmpl	C		compare two string, ignoring case
strcpy	S	Small	copy a string
strcspn	C		locate character not in given set
strdup	C		duplicate string, using malloc
strlen	S	Small	return string length
strlwr	C		convert string to lower case
strncat	S	Small	concatenate to a given maximum length
strncmp	S		compare to a given maximum length
strncpy	S	Small	copy to a given maximum length
strpbrk	C		get pointer to break character
strrchr	C		locate character from end of string
strrev	C		reverse string
strset	C		initialise string with given character
strspn	C		locate character in given set
strtok	C		look for token using given delimiters
strupr	C		convert string to upper case

Character classification

isalnum	S	Macro	test for a letter or digit
isalpha	S	Macro	test for a letter
isascii	S	Macro	test for ASCII value less than 128
iscntrl	C	Macro	test for a control character (less than 32 or 127)
isdigit	S	Macro	test for a decimal digit
isgraph	C	Macro	test for a printing character and not space
islower	S	Macro	test for a lower case letter
isprint	S	Macro	test for a printable character
ispunct	S	Macro	test for a punctuation character
isspace	S	Macro	test for a space, tab, return, line or form feed
isupper	S	Macro	test for an upper case letter
isxdigit	C	Macro	test for a hexadecimal digit

Note: these are also supplied as functions in the full library.

Conversion functions

atof	S		convert string to double
atoi	S		convert string to integer
atol	S		convert string to long
ecvt	C		convert floating point number to E format
fcvt	C		convert floating point number to F format
gcvt	C		convert floating point number to G format
strtod	C	Small	convert string to double
strtol	C	Small	convert string to long
toascii	S	Macro	force character to ASCII (in range 0 to 127)
tolower	S	Macro	convert character to lower case
toupper	S	Macro	convert character to upper case

Note: the last three macros are also supplied as functions in the full library.

Memory functions

memchr	C	locate character in memory
memcmp	C	compare memory blocks
memcpy	C	copy memory block
memset	C	initialise memory contents
movmem	C	move memory block, checking for overlapping

Memory allocation functions

calloc	S		allocate and initialise block of memory
free	S	Small	free previously allocated memory block
mallinfo	A	Small	return memory status information
malloc	S	Small	allocate block of memory

Mathematical functions

acos	C	Maths	inverse cosine
asin	C	Maths	inverse sine
atan	C	Maths	inverse tangent
atan2	C	Maths	inverse tangent of y/x
ceil	C	Maths	lowest integer > = number
cos	C	Maths	cosine
cosh	C	Maths	hyperbolic cosine
exp	C	Maths	exponential
fabs	C	Maths	absolute value of double
floor	C	Maths	highest integer < = number
fmod	C	Maths	floating point modulus
frexp	C	Maths	split into mantissa and exponent
ldexp	C	Maths	construct number from mantissa and exponent
log	C	Maths	natural log
log10	C	Maths	log to base 10
matherr	C	Maths	maths error handling function
modf	C	Maths	whole and fractional part
pow	C	Maths	calculate x to the power of y
rand	C	Maths	random number generator
sin	C	Maths	sine
sinh	C	Maths	hyperbolic sine
sqrt	C	Maths	square root
srand	C	Maths	seed the random number generator
tan	C	Maths	tangent
tanh	C	Maths	hyperbolic tangent

Miscellaneous functions

abort	C	Small	terminate program execution
abs	S	Macro	absolute value
execv	C	Small	execute another program using argument vector
exit	S	Small	exit program and close files
longjmp	S	Min	long jump to a given location
max	C	Macro	maximum of two values
min	C	Macro	minimum of two values
setjmp	S	Min	set location for jump
_exec	S	Min	execute another program
_exit	S	Min	exit program

System functions

bdos	C	Small	call CP/M BDOS function
bios	A		call CP/M BIOS function
call	A		call a machine code subroutine
drsearch	A	Min	look for file on all drives
filesize	C		find the size of a file
firmware	A		call firmware routine
getdrive	A		get the currently selected drive
inp	C		input from a port
outp	C		output to a port
peek	C		read byte from memory address
poke	C		store byte at memory address
rename	C		rename file
seldrive	A		change currently selected drive
setfcb	A	Small	set up file control block for BDOS function
settime	A		set the time
time	A		get the time
unlink	S	Small	delete file
version	A	Min	determine model of computer being used
_getlim	A	Min	get lower limit of free memory
_getsp	A	Min	get upper limit of free memory
_putlim	A	Min	set lower limit of free memory

APPEN 2-8

A3. COMPILER ERROR MESSAGES

These error messages are displayed on the screen when errors occur. They are contained in the file 'ERRMSG.R'. If this file is not present, error numbers will be displayed as listed below.

1 '\0' not allowed in strings
2 Cannot open file %s in mode %s
3 Out of memory
4 End of file encountered before #endif
5 End of comment before start of comment
6 End of file found before end of comment
7 " or ' expected
8 Macro with wrong number of parameters
9 Illegal #include (use "filename" or <filename>)
10 #else without a #if
11 #endif without a #if
12 #line has to be followed by an integer expression
13 Illegal # directive
14 Macro pool overflow in lexical analyser
15 Illegal #if or #assert expression (does not reduce to a constant)
16 Need '(' to precede parameters in a macro
17 Macro parameters not closed with a ')'
18 '{' expected to start array initialisation
19 '{' expected to start structure initialisation
20 '}' expected to end initialiser list
21 '}' expected after single value initialiser
22 Expression does not reduce to constant
23 Cannot initialise unions
24 Initialisation string is too long
25 Multiply defined global identifier %s
26 '{' expected at the start of function body
27 Label reference(s) not resolved at the end of function
28 Multiply defined local identifier %s
29 Conflicting type declarations
30 Declarator buffer overflow
31 Cannot define a function here
32 Warning: Size of object is zero/undefined
33 Missing ')' in declarator
34 Cannot have an identifier in an abstract declarator
35 Code generation stack overflow : Function too complex
36 Badly defined declarator
37 Filename already given on command line
38 Cannot have an array of functions

39 No ')' in function declarator or declaration
40 Illegal function declarator
41 Missing ']' in array bound
42 Unexpected value in function declarator
43 Structure/union forward declared but has not been defined yet
44 Illegal type of actual parameter
45 Typespec tag already exists
46 Expression required
47 RHS of operator expected
48 Illegal pointer operation
49 Warning: LHS of comma ignored
50 Mismatched pointers
51 Error in x?y:z
52 Sizeof argument required
53 Size of aggregate not known
54 Bad arg for unary &
55 Filename too long
56 Bad argument for + +
57 Bad argument for unary -
58 Bad argument for ~
59 Bad argument for !
60 Bad argument for unary *
61 Bad argument for cast
62 #assert failed
63 %s undeclared variable - assumed int
64 Unknown LHS for . or - >
65 Unknown offset into structure or union
66 Parameter list too complex
67 Illegal formal parameter in macro definition
68 Subscript required after '['
69 Expression workspace overflow
70 Symbol table full
71 Prefix for '[' must be array or pointer
72 Actual parameter required
73 ')' or ',' required after actual parameter list
74 Argument for * must be a pointer
75 l-value required
76 Expression cannot be converted to boolean
77 Illegal type for operand
78 Illegal type coercion
79 Error in post-processor files = 0x%.4x
80 Use compile <file> [-d<macro>] [-f] [-m<number>] [-q] [-l<text>]
81 Type specifier expected
82 %s tag already declared
83 Structure or union specifier expected

84 Option %c not understood in command line
85 %s multiply defined in aggregate
86 '}' in structure declaration
87 Fields not supported
88 Cannot declare a function here
89 Semi-colon expected
90 %s doubly declared as parameter
91 %s is an user type - bad parameters
92 %s is not in the formal parameter list
93 Symbol table overflow
94 Character 0x%.2x not understood by lexical analyser
95 Doubly defined label
96 Identifier expected after a goto
97 While expected to close do loop
98 '(' expected after a for statement
99 ')' expected after a for statement
100 Statement or declarator in invalid context
101 '(' expected before condition
102 ')' missing from condition
103 Cannot define a function as external
104 Illegal control character inside string or character constant
105 Error %d while accessing link output file
106 Colon expected after case
107 Colon expected after default
108 End of file found unexpectedly (missed " earlier?)
109 Error %d while accessing temporary data file
110 End of file before end of compound statement
111 %s is illegally initialised
112 Warning: Condition is constant in x?y:z
113 ')' expected immediately after identifier in defined
114 ')' expected
115 Run out of global table
116 Non-portable implicit pointer conversion performed
117 Too many macro arguments

A4. SUMMARY OF EDITOR COMMANDS

This summary is divided into the following categories and gives concise details of the syntax of all commands used by the editor.

(a) Edit mode commands
(b) Command mode commands
(c) External utility program commands

(a) Edit mode commands

PCW8512	PCW8512	CPC6128	
Cursor Movement			
ALT-CHAR	←	←	Move left one character
CHAR	→	→	Move right one character
	↑	↑	Move up one character
	↓	↓	Move down one character
ALT-WORD	SHIFT-←	SHIFT-←	Move left one word
WORD	SHIFT-→	SHIFT-→	Move right one word
	SHIFT-↑	SHIFT-↑	Scroll back 1 line
	SHIFT-↓	SHIFT-↓	Scroll forward 1 line
LINE	ALT-←	CTRL-←	Move to start of line
EOL	ALT-→	CTRL-→	Move to end of line
	ALT-↑	CTRL-↑	Scroll back 18 or 25 lines
	ALT-↓	CTRL-↓	Scroll forward 18 or 25 lines
	ALT-Q	CTRL-Q	Scroll back one screen (no overlap)
	ALT-Z	CTRL-Z	Scroll forward one screen (no overlap)
	ALT-[	CTRL-[	Move to start of memory
	ALT-]	CTRL-]	Move to end of memory
ALT-DOC	ALT-[ALT-[	CTRL-[CTRL-[	Move to start of document
DOC	ALT-] ALT-]	CTRL-] CTRL-]	Move to end of document
ALT-PARA	ALT-<	CTRL-<	Move back one paragraph
PARA	ALT->	CTRL->	Move forward one paragraph
		CTRL-£	Move to top left of screen
	ALT-L	CTRL-L	Go to last position
	ALT-G	CTRL-G	Go to line/column

ENTER	RETURN	RETURN	Insert mode - split line and move to start of next line Overwrite mode - move to start of next line
	SHIFT-RETURN	SHIFT-RETURN	Move to column 1 of next line
	ALT-RETURN	CTRL-RETURN	Move to column 1 of next line
	TAB	TAB	Insert mode - insert tab character Overwrite mode - move to next tab
	SHIFT-TAB	SHIFT-TAB	Insert mode - move to next tab Overwrite mode - insert tab character

Insertion and deletion

	ALT-I	CTRL-I	Insert line
ALT-CAN		CTRL-CLR	Delete line
	ALT-←DEL		Delete to start of line
ALT-E	ALT-DEL→	CTRL-E	Delete to end of line
	←DEL	DEL	Delete character before cursor
	DEL→	CLR	Delete character at cursor
	SHIFT-←DEL	SHIFT-DEL	Delete word left
	SHIFT-DEL→	SHIFT-CLR	Delete word right
	ALT-TAB	CTRL-TAB	Toggle insert/overwrite mode
	ALT-A	CTRL-A	Transpose (alternate) characters
	ALT-U	CTRL-U	Undo last delete operation
	ALT-*	CTRL-*	Split line at cursor
	ALT-+	CTRL-+	Join lines
SHIFT-ALT-CAN			Clear text

Block commands

SHIFT-⊞	SHIFT-COPY	SHIFT-COPY	Set or clear block markers
CAN	ALT-K	CTRL-K	Clear all block markers
PASTE	ALT-M	CTRL-M	Move block
COPY	ALT-COPY	CTRL-COPY	Copy block
CUT	ALT-CUT	CTRL-DEL	Delete block
	ALT-O	CTRL-O	Copy block from other document

Find and replace, place markers

FIND			Find string
EXCH			Replace string
⊞		COPY	Next find
⊟	ALT-@ @	CTRL-@ @	Previous find
	ALT-@ n	CTRL-@ n	Set/Go to marker (0 to 9)
	ALT-@ [	CTRL-@ [	Go to [block marker
	ALT-@]	CTRL-@]	Go to] block marker
UNIT	ALT-⊞	CTRL-@ +	Go to next marker
ALT-UNIT	ALT-⊟	CTRL-@ -	Go to previous marker

Other commands

	ALT-H	CTRL-H	Help on/scroll forwards
SH-ALT-H	ALT-V B	CTRL-V B	Help scroll backwards
	ALT-V H	CTRL-V H	Edit help off
	ALT-V S	CTRL-V S	Hard spaces visible/hidden
	ALT-V T	CTRL-V T	Tabs and returns visible/hidden
	ALT-V cursors	CTRL-V cursors	Insert arrow symbols
	ALT-Y	CTRL-Y	Switch between documents in memory
	ALT-/	CTRL-/	Convert to upper case
	ALT-.	CTRL-\	Convert to lower case
	ALT-space	CTRL-space	Insert a space
	EXTRA-letter	CTRL-0 letter	Insert stored phrase
PTR			Access internal printer controls
EXTRA-PTR			Screen dump (internal printer)
ALT-ENTER		CAPS LOCK	Caps lock
SHIFT LOCK	CTRL-CAPS LOCK		Shift lock
EXIT	STOP	ESC	Exit from edit mode

Extra characters

SHIFT-ALT-↑	CTRL-V ↑	Up arrow
SHIFT-ALT-↓	CTRL-V ↓	Down arrow
SHIFT-ALT-←	CTRL-V ←	Left arrow
SHIFT-ALT-→	CTRL-V →	Right arrow

PCW8256/8512 extra characters

EXTRA-.	¦
EXTRA-@	\
EXTRA-:	↑

(b) Command Mode

The following commands are listed according to a number of categories and alphabetically within the category for ease of location. The categories are:-

Document handling
Text manipulation
Printer control and printing
Disc Drive selection and cataloguing
Disc file manipulation
File protection
Phrase, Exec and Symbol commands
Miscellany

Key to parameters:

a	an ASCII character
dr	drive letter
f	a filename (may include drive)
af	ambiguous filename (may contain wildcards)
newf	new filename
oldf	old filename
n	an integer between 0 and 255
nn	an integer between 0 and 65535
(x)	an optional parameter

Items marked with '<..>' are mandatory.

Command		Abbr.	Description
Text file handling			
CLEAR	-		Clear text.
LOAD	<f>	L	Load new file.
MERGE	<f>	MER	Merge file with current text.
NAME	<f>	N	Assign name to current file.
SAVE	<f>	S	Save text.
SAVEB	<f>	SB	Save block.
SPOOL	<f>	SPON	Echo all screen output to a file.
SPOOLOFF		SPOFF	Turn off echo to file.
SWAP		SW	Swap between two documents in memory.
TYPE	<f>	T	Display file contents on screen.

Text manipulation

FIND		F	Find string.
GOTO	<a><nn>	G	Goto Line/Column number.
REPLACE		R	Find and replace string.
NUMBER		NUM	Number lines of text.
NUMBERB		NUMB	Number lines of text in marked block only.
TAB	(col)...(col)	-	Set tabs.
CAN or CLR key			Clear screen.

Printer control and printing

BM	<n>	-	Bottom margin for printing.
INTERNAL		INT	Select PCW Printer.
PARALLEL		PAR	Select parallel (Centronics) printer.
PL	<n>	-	Page length for printing.
PRINT	(f)	P	Print file, optionally from disc.
PRINTB		PB	Print marked block of text only.
PRINTER	<f>	PR	Load printer driver.
PRINTON		PRON	Echo all screen output to printer.
PRINTOFF		PROFF	Turn off echo to printer.
SERIAL		SER	Select serial printer port.

Disc Drive selection, cataloguing and disc formatting

A:		A	Select drive A.
B:		B	Select drive B - valid if drive installed.
C:		C	Select drive C - valid if drive installed.
D:		D	Select drive D - valid if drive installed.
M:		M	Select drive M - (PCW only).
DFORM	-		Format disc. CF2 or CF2DD format.
DFORMD	-		Format disc. CPC6128 Data format.
DRIVE	<dr>	DR	Select drive. - Any drive A-P if installed.
GROUP	<n>	GR	Select group/user number.
USER	<n>	U	Select group/user number.
CAT	(dr)or(grp)or(af)	DIR	Catalogue files. Opt. drive, group, name.
INFO	<af>	-	File information.

Disc file manipulation

COPY	<oldf><newf>	-	Copy file, creating backup.
COPY	<af>(<grp>)(<dr>)		Copy file(s). Wildcards permitted.
ERACOPY	<oldf><newf>	ECOPY	Copy file, erasing old version.
ERACOPY	<af>(<grp>)(<dr>)		Copy file(s), erasing old. Allows wildcards.
ERASE	<af>	ERA	Erases file(s). Wildcards permitted.
RENAME	<newf><oldf>	REN	Rename file.

File protection

ACCESS	<af>	ACC	Set file(s) to read-write status.
PROTECT	<af>	PROT	Set file(s) to read-only status.

Phrase, Exec and Symbol commands

EXEC	<f>	X	Execute file of commands.
LPHRASES			LP List defined phrases.
PHRASE		KEY	Define phrase or function key.
SYMBOL	<n><n1>...<n8>	SYM	Re-define a screen character.

Miscellany

HELP	(subhead)	H	Display HELP subheadings (opt. param in editor only).
PAUSE	-		Pause (optionally with message).
QUIT		Q	Quit the editor.

COPY or (PCW only) ⊞ key — Recall last command typed in.

(c) External utility program commands

The following commands call programs which must be available to the editor. In other words the specified '.COM' files must be on one or other of the discs. These programs are specially configured so that when completed, control returns to the editor without loss of the original text (See note below). Any program may be called using the * prefix, and any text in memory will be saved to a temporary file the name of which will be passed as a parameter, but unless specifically written to work with the editor, control will not automatically return to the editor. Typing APED from the CP/M command prompt will re-load the editor, with any text in memory intact (See note below for single drive CPC6128).

Note: CPC 6128 users with only one disc drive and PCW users who are not using drive M as the 'Temporary' drive should ensure that any documents are saved before using these options, as use of these external programs will usually require the changing of discs in order to run the utility program. On return to the editor, the document can be reloaded in the normal way.

AC	-	Compile, link and run C program.
ASM	MA	Assemble m/c program
CONFIG	-	Editor configuration utility program.
DCOPY	-	Copy single sided disc. Utility program.
MM	-	Maxam II Monitor (full version).
MON	-	Maxam II Monitor (whichever version present).
MSM	-	Maxam II Monitor (small version).
RUNC	-	Enter C run time system.
SETPRINT	SP	Printer driver creation utility.
*<progname>	-	Call a separate program, automatically passing existing text file as a parameter.
*<progname><filename>	-	Call a separate program with given filename.
*<progname>""	-	Call a separate program without passing a filename.
*<drive>:<progname>	-	Call a separate program from specified drive.

APPEN 4-8

A5. KEY TRANSLATIONS

These are the codes which must be used in a phrase definition to string command sequences together.

PCW	CPC	code	PCW	CPC	code
ALT-@	CTRL-@	0	ALT-[	CTRL-[	27
ALT-A	CTRL-A	1	ALT-.	CTRL-\	28
ALT-B	CTRL-B	2	ALT-]	CTRL-]	29
ALT-C	CTRL-C	3	-	CTRL-£	30
ALT-D	CTRL-D	4	-	CTRL-0	31
ALT-E	CTRL-E	5	ALT-<	CTRL-<	218
ALT-F	CTRL-F	6	ALT->	CTRL->	219
ALT-G	CTRL-G	7	ALT-(	CTRL-(	220
ALT-H	CTRL-H	8	ALT-)	CTRL-)	221
ALT-I	CTRL-I	237	ALT-*	CTRL-*	222
ALT-J	CTRL-J	10	ALT-+	CTRL-+	223
ALT-K	CTRL-K	11	ALT-/	CTRL-/	231
ALT-L	CTRL-L	12	ALT-hyphen	CTRL-hyphen	227
ALT-M	CTRL-M	238	ALT-space	CTRL-space	235
ALT-N	CTRL-N	14			
ALT-O	CTRL-O	15	↑	↑	240
ALT-P	CTRL-P	226	↓	↓	241
ALT-Q	CTRL-Q	17	←	←	242
ALT-R	CTRL-R	18	→	→	243
ALT-S	CTRL-S	19	SHIFT-↑	SHIFT-↑	244
ALT-T	CTRL-T	20	SHIFT-↓	SHIFT-↓	245
ALT-U	CTRL-U	21	SHIFT-←	SHIFT-←	246
ALT-V	CTRL-V	22	SHIFT-→	SHIFT-→	247
ALT-W	CTRL-W	23	ALT-↑	CTRL-↑	248
ALT-X	CTRL-X	24	ALT-↓	CTRL-↓	249
ALT-Y	CTRL-Y	25	ALT-←	CTRL-←	250
ALT-Z	CTRL-Z	26	ALT-→	CTRL-→	251
TAB	TAB	9	RETURN	RETURN	13
SHIFT-TAB	SHIFT-TAB	228	SHIFT-RETURN	SHIFT-RETURN	236
ALT-TAB	CTRL-TAB	225	ALT-RETURN	CTRL-RETURN	236
DEL→	CLR	16	ALT-CAN	CTRL-CLR	230
SHIFT-DEL→	SHIFT-CLR	229	ALT-CUT	CTRL-DEL	232
ALT-DEL→		5	ALT-COPY	CTRL-COPY	234
←DEL	DEL	127	⊞	COPY	224
SHIFT-←DEL	-	211	STOP/EXIT	ESC	252
ALT-←DEL	-	212	Enter edit mode		253
SHIFT-COPY	SHIFT-COPY	233	Enter command mode		254

APPEN 5-2

A6. SYSTEM ERROR MESSAGES

This section lists error messages relating mainly to disc operation. These are termed system error messages because they do not relate to any particular program or command, and may occur at any time.

Disc missing or read fail - Retry Ignore or Cancel?

(i) The disc being used has been taken out of the drive. Insert the disc and press R.

(ii) The disc is not formatted.
Press C to cancel the operation. The disc must be formatted using DFORM before it can be used.

(iii) PCW only. This error is given if a CF2DD disc is put into drive A, or the wrong way round in drive B.

(iv) The disc may be faulty or corrupted. Press R to retry. If the error persists try re-formatting the disc.

Drive not ready - Retry Ignore or Cancel?

The disc being used has been taken out of the drive. Insert the disc and press R.

Disc error - Retry Ignore or Cancel?
Seek fail - Retry Ignore or Cancel?
Data error - Retry Ignore or Cancel?
No data - Retry Ignore or Cancel?
Missing address mark - Retry Ignore or Cancel?
Media changed - Retry Ignore or Cancel?
Media change occurred - Retry Ignore or Cancel?

If any of these errors occur the disc may be faulty or corrupted. Press R to retry. If the error persists try re-formatting the disc.

Write protected - Retry Ignore or Cancel?

The write protect tab is pushed in. Remove the disc, slide the tab out, and press R.

Disc unsuitable for drive - Retry Ignore or Cancel?

PCW only. The disc in drive B is formatted as a CF2 disc, and so may only be written to in drive A.

Bad format - Retry Ignore or Cancel?

The disc is not formatted, or is of a non-Amstrad format, or the disc may be faulty.

File is read only

The chosen file cannot be written to because it has been protected. Use ACCESS to unprotect the file.

Directory full

The maximum number of files allowed on a disc has been reached. There may still be room on the disc, so any unwanted files should be deleted.

Disc full

The storage capacity of the disc has been reached. Often this can be remedied by erasing backup files. Type ERASE *.BAK or press ALT-f7 (PCW) or CTRL-f9 (CPC6128).

File not found
File does not exist

There is no file of the chosen name on the disc. Check that the name was typed correctly, that the correct disc is in the drive, and that the correct drive is selected.

Bad filename

The combination of characters chosen as a filename is not allowed. Valid names consists of up to 8 characters, followed optionally by a full stop and an extension of up to 3 characters. Certain characters are not allowed.

Maximum number of files open

There is a limit to the number of files that can be open at the same time. In normal use this limit will never be reached.

Insufficient memory for program

The program has used all the available computer memory. This error will not occur in normal use.

EXEC file read error

A disc error occurred when reading commands from an exec file. This could be caused by removing the disc containing the exec file, or by a faulty disc.

This program will only run under Amstrad CP/M Plus

Arnor CP/M Plus programs will only work on Amstrad computers with CP/M Plus. In particular they will not run on other CP/M systems. This is because special use is made of Amstrad specific features to attain the best performance.

If this message occurs when using an Amstrad computer, which is possible if some other software is installed, all may not be lost. Contact Arnor for help.

APPEN 6-4