***** C POWER™ *****

by Brian Hilchie

A C Language Compiler Package
for the Commodore 64®

A Product Of Pro-Line Software Ltd.,

Manufactured in CANADA by:

PRO-LINE SOFTWARE LTD.,
755 The Queensway East, Unit 9,
Mississauga, Ontario, CANADA, L4Y 4C5.
(416) 273-6350

Designed for use with the Commodore 64® Computer and one or two Commodore or MSD disk drives.

4th Printing October 1, 1985

Foreword

The C programming language is probably one of the more important developments in the micro computer field because it is not tied to any one particular manufacturer's hardware or system. Thus it is rapidly becoming a major program transportability factor between the multitude of different CPU's and operating systems, from mainframes to micros. C was originated and placed in the public domain by Dennis M. Ritchie, of Bell Labs.

The C language has been described as a "simple and elegant" programming language with an absence of restrictions and a generality that make it far more effective and convenient than "supposedly more powerful programming languages". The UNIX operating system was written in C. Most current terminal programs and many of the new word processors have been written in the C language. What an incredible luxury for a programmer to be able to compile his C source for each of several different CPU's instead of rewriting new source code for each.

C is easy to learn, as it has only about a dozen commands and depends on function libraries to gain speed and efficiency.

Welcome to the future.

CONTENTS

Foreword	i
Getting Started	ii
Compatibility	1
ML Interface	5
Coding Efficiently	6
C SHELL, Command Interpreter	7
EDITOR	9
COMPILER	10
LINKER	10
ANOMALIES	11
LIBRARY, Introduction	12
FUNCTION, Index	13
FUNCTION, Library	14
WARRANTY POLICIES	38
C SHELL UTILITIES	39

GETTING STARTED

Side one of your C POWER System Disk contains the C Shell, Editors, Compiler, Linker, Stdio library, Math library and three sample programs: sort.c, find.c and wfreq.c. Of these programs, only the Compiler program is protected from copying and we suggest that a back-up system disk be made and used for all functions except actual compiling.

Side two, the back side, of your System Disk contains the Stdlib.l and Syslib.l Function libraries. This side of the disk may be backed-up and we suggest that you make and use a work copy of the Library Disk rather than use the original.

A C POWER bulletin board has been established so that any upgrades, extra functions and function libraries can be made easily available, and assistance can be supplied by our C language experts. The CBBS 24 hour number is (416) 276-6811.

Compatibility

The Pro-Line C POWER compiler is generally quite compatible with those found on other microcomputers and mainframes. However, there are a few potential compatibility problems which should be kept in mind if you plan to port code from other machines to the C-64 or vice versa. The main areas of concern are:

1. Standard C features not implemented or not implemented completely
2. Implementation dependant details (e.g. type sizes)
3. Non-standard library functions

These will be discussed in the following sections.

1. Features Not Implemented

1.1 Bit Fields

To obtain the effect of bit fields the programmer must use shifts and masks. For example, to get the value of the third through sixth bits of an int variable y one might use the code

```
x = (y >> 2) & 0xf;
```

To assign a quantity to the same field one might use

```
y = (y & <tilde>0x3c) | (x << 2);
```

where <tilde> is the bitwise not operator. This is admittedly awkward, but in practice bit fields are not frequently used so their absence should not present a great problem.

1.2 Pointer Initialization

Static pointer variables may not be initialized except for character pointers initialized with strings. For example, the declaration

```
static char *s = "this is a string";
```

is allowed while

```
static int x;
static int *y = &x;
```

is not allowed. Static variables include variables explicitly declared static and variables declared outside of a function but given no explicit storage class declaration. Auto variable initialization is fully supported.

1.3 Conditional Operator Type Conversions

The second and third operands of the conditional operator are not brought to a common type if they are different. If the types are different the programmer must make the conversion explicit. For example the code

```
int i;
float f, x;

x = a==b ? f : i;
```

may generate incorrect machine code, while the statement

```
x = a==b ? f : (float) i;
```

will generate correct machine code.

1.4 Operators

Certain operators under certain conditions will not work unless the expressions containing them are parenthesized. Namely the logical or, conditional, assignment, and comma operators behave this way in subscripts and constant expressions. For example, the expression

```
a[i > j ? i : j]
```

will generate a syntax error, while the expression

```
a[(i > j ? i : j)]
```

will be accepted.

2. Implementation Details

2.1 Type Sizes

The following table lists the size in bytes of all data types supported by the compiler:

Type	Size
char	1
short	2
int	2
long	2
unsigned	2
float	5
double	5
pointer	2

Note that short, int, and long are synonyms, as are float and double.

Integers and pointers are stored low byte first, high byte last. Float quantities are stored in the same format as BASIC variables.

2.2 Identifiers

All identifiers (including external identifiers) are significant to eight characters, except for goto labels, which are significant to seven characters. Identifiers may contain up to 255 characters.

2.3 Sign Extension and Sign Fill

The compiler does not sign extend character quantities. When characters are converted to integers, the high byte is simply set to zero.

Similarly, integers are not sign filled when shifted right. Vacated bits are set to zero even if the quantity being shifted is negative.

2.4 Register Declarations

All register declarations are ignored. However, the first 32 bytes of integers, characters, and pointers declared in a function are placed in the zero page, while the remainder are put in a stack in main memory. Thus if certain variables are used frequently they should be declared before those used less frequently.

2.5 Character and String Constants

Multi-character constants are not supported. If more than one character appears in a character constant only the first is taken.

String constants may not contain more than 255 characters.

2.6 Header Files

The preprocessor lines

```
#include <filename>
```

and

```
#include "filename"
```

are synonymous. The header file filename must be on the same disk as the source file containing the preprocessor line. If the file is one of the standard header files such as stdio.h, it must be copied from the system disk to your work disk.

2.7 Program Size

The compiler may not be able handle some large source files. If an overflow message is printed when you are compiling a large program don't despair; split the program into two or more smaller source files and compile them separately.

If the linker gives an overflow message you have two possible courses of action: use more efficient coding techniques or rewrite some of the program in assembler (see the section on interfacing C with assembler).

2.8 EOF Marker

End Of File may be signaled from the keyboard with a period (.) on an otherwise blank line.

3. The Library

Although an attempt has been made to make the library provided with the compiler as standard as possible, the lack of a true operating system on the Commodore 64 has required that some compromises be made. Before using a function listed in the UNIX (or other) standard library look up the description of the Pro-Line version in the library section.

Interfacing with Machine Language

Although C is well suited to a wide range of tasks, no high level language can match the speed and compactness of hand written assembly code, especially with a microprocessor such as the 6502. Thus if speed is critical for a particular problem, or if a program is too large to be written entirely in C, it would be desirable to be able to call machine language subroutines from within a C program. This can be done with the library function sys, whose description can be found in the library section.

The recommended procedure is to first write those sections of the program which are to be eventually written in assembler as C functions. Once the program is debugged, the C functions may be replaced one at a time with assembler routines (any assembler may be used). The advantage of this approach is that it maximizes portability since the very non-portable machine language calls are isolated in a set of functions. Also, if the computer the program is being ported to is more powerful than the C-64, machine language routines may be unnecessary, in which case you can use the original C functions.

The following small example illustrates the procedure. The code which is to be written in assembler is to add three small positive integers. The code is first written as a C function add3, which is shown below along with a driver program:

```
#include <stdio.h>

main()
(*
    int a, b, c;

    while ((printf("enter three numbers: "),
            scanf("%d %d %d", &a, &b, &c)) != EOF)
        printf("sum: %d", add3(a, b, c));
*)


add3 (arg1, arg2, arg3)
(*
    return (arg1 + arg2 + arg3);
*)
```

The rewritten add3 with the associated machine language subroutine (located at c000 hex) is shown below:

```
add3 (args)
float args;
(*
    char a, x, y;

    x = (int) &args;
    y = (int) &args >> 8;
    sys (0xc000, &a, &x, &y);
    return a;
*)
```

```
c000 stx $4b
     sty $4c
     clc
     ldy #0
     lda ($4b),y ; load arg 1
     ldy #2
     adc ($4b),y ; add arg 2
     ldy #4
     adc ($4b),y ; add arg 3
     rts
```

Note that only one argument is declared and that it is given type float. This is to ensure that the arguments remain in consecutive memory locations and in the correct order. Note also that the x and y registers are loaded with the low byte and high byte of the address of the argument list respectively and that the result is returned in the accumulator.

If the machine code is placed in higher memory than the C code, the library function highmem should be called to limit the memory which the C code can use. In the above example, the statement

```
highmem (0xc000);
```

should be included in the main function before the first call to add3.

If the machine code is to placed below the C code remember to enter an appropriate starting address when linking the C program.

Zero Page Usage

The zero page locations 20-33, 43-74, and 253-254 decimal are reserved for permanent system use and should not be disturbed. The locations 34-42, 75-96, and 251-252 are used as temporary storage and may also be used as such in your machine language routines. The remainder of the zero page may be used as you wish, but caution should be observed when using locations used by kernal routines or BASIC floating point routines.

In addition, the tape buffer (33c-3fb hex) is used by the system and should not be disturbed.

Coding Efficiently

Due to the way certain features have been implemented in the compiler and the nature of the 6502 instruction there are a few non-obvious ways of improving the efficiency of some programs written with C POWER:

Declare variables unsigned rather than int if they never contain negative values. For example, variables which are used only as array indices should always be declared unsigned.

If a function contains many variables, declare those used most frequently first.

C SHELL, COMMAND INTERPRETER

The first program on the System Disk, and the first one you should load and run, is a "Shell" mini command interpreter. The Shell is a program that supports command line arguments and I/O redirection along with the compiler and other programs designed to work under it. Load and run the program "shell" and you will soon see a dollar sign prompt on the screen waiting for one of the commands listed below.

The Shell lets you define a work disk and a system disk. The defaults for both are device 8 and drive 0, but they may be changed as described below if you are lucky enough to have two disk drives. There are a number of built-in commands which are also listed here. Items in square brackets are optional. (Arguments may optionally be enclosed in double quotes, which is necessary if they contain spaces.)

bye
Exit to BASIC.

l [pattern]
List the directory of the work disk to the standard output (screen).

ls [pattern]
List the directory of the system disk to the standard output (screen).

rm [filename]
Remove (scratch) a file on the work disk.

mv [file1 file2]
Move (rename) file1 to file2 on the work disk.

pr [filename]
List the contents of a file on the work disk to the standard output (screen).

pr >> [filename]
Same as above but redirects the output to device 4, (usually the printer).

disk [string]
Send a string to the work disk device. ie: disk n0:[header],[id] would format the disk in drive 0.

load [command]
Load, but don't run, the specified command from the work disk or the system disk, wherever it is.

work [device# drive#]
With no arguments, work shows the current work disk device and drive numbers. It may also be used to change the device and drive numbers.

sys [device# drive#]
Same as above except for the system disk.

There are a number of commands provided which are loaded from disk:

ed [filename]
ced [filename]
Run the editor (ed) or syntax checker (ced). If a file is specified it is loaded into the main editing buffer.

cc [-p] [filename.c]
Compile the C program in the specified file. If the -p option is specified the compiler will assume that both the source/object and compiler disks are present and will not tell you to change disks all the time. The work drive is assumed to contain the source/object disk and the system drive is assumed to contain the compiler.

link [-s [address]]
Run the linker. If no arguments are given programs are linked in such a way that they will run under the Shell. In this case the program names must end with ".sh". The -s option indicates that programs are to be linked so that they will run independantly of the Shell. If no address is specified the programs will be linked at the start of BASIC memory so that they be LOADed and RUN. If an address is specified in either decimal or hex (hex numbers must be identified by a preceding $ symbol), the programs will be linked there. Object files will be taken from the work disk, and the library will be looked for on the system disk.

Programs written in C may access command line arguments in the same manner as described in C PRIMER PLUS starting on page 407. Also, I/O may be redirected to and from disk files (on the work disk) like UNIX as described on page 152. ">>" will direct the standard output to device 4 (usually a printer).

A few programs (adapted from examples in "The C Programming Language" book by Kernighan & Ritchie) are provided to illustrate these features:

sort [-n]
Sorts the standard input and writes the result to the standard output. The -n option indicates that the input is to be sorted in numerical order. The default is alphabetical order (actually, lexicographic order).

find [-x] [-n] string
Finds all occurences the specified string in the standard input. If the -x option is specified, only lines NOT containing the string are output. If the -n option is specified output lines will be numbered.

wfreq
Counts the number of occurences of each word in the standard input. A word is defined as a string of characters beginning with a letter and followed by up to 19 letters and digits.

As stated above, programs that work with the Shell mini command interpreter must have file names ending with ".sh". These programs may be invoked by merely typing the name without the ".sh" part along with any arguments and I/O redirection specifiers. The work disk will be searched first, and if the program is not there the system disk will be searched. Once a program is loaded it may be run any number of times without re-loading by using the built in run command.

EDITOR (ed.sh)

At the Shell program prompt ($), type in ed filename <return> or, if you want the syntax checking editor ced filename <return> and the chosen editor will be loaded and run. If a filename is specified, it will be loaded automatically from the editor.

Edit Mode, Function Keys
crsr keys - up, down, left and right
f1, f2 - page down, page up
f3, f4 - search down, search up
f5 - cut text (<SHIFT><RUN/STOP> starts, use cursor keys to set range
f6 - paste text (f5 deletes, f6 inserts
f7 - go to end of line
f8 - go to start of line
<clr/home> - go to bottom of buffer
<shift> <clr/home> - go to top of buffer
<shift> <return> - open a line
<shift> <run/stop> - enter select mode for cut
<run/stop> - enter command mode

Command Mode
In command mode the following commands may be entered:

dir - list disk directory
disk [string] - send the optional string to the disk drive
GEt filename - reads file from disk
PUt filename - writes file to disk
PRint - dumps contents of current buffer to device 4
GOto buffername - change current edit buffer
List - lists buffers currently in use
CHeck - (ced.sh only) - check syntax
QUIT - exit editor
CLEAR - clear current buffer
CLEAR buffername - clear specified buffer
/searchstring<f3> - set search string and search down
/searchstring<f4> - set search string and search up
/dog - search for next occurence of 'dog'
/dog/cat - search for next 'dog' and replace it with 'cat'
/dog/cat/ - search and replace all 'dog's with 'cat's (needs f2, f3 above)
/dog/ - delete next occurence of 'dog'
/dog// - delete all occurences of 'dog'
<return> - return to edit mode

Special Symbols used in C
The special C characters may be obtained as follows:

curly brackets: <shift> + and <shift> -
back slash: <english pound sign>
tilde: <logo> p
underscore: <logo> @
vertical bar: <logo> *

NOTE: Use the <logo> key in the above examples the same as you would a shift key ... hold it down while pressing the next character.

COMPILER (cc.sh)

At the Shell program prompt ($) type cc [-p] filename.c <return>, and the compiler will be loaded and run with the specified filename.c. If you type in the -p prior to the filename.c, the compiler will assume you are using two disk drives. The filename should always end with ".c". When finished, the compiler will leave an object file on disk with the same name as the source except the extension ".c" will be replaced with ".o".

LINKER (link.sh)

At the Shell program prompt ($), type in link [-s [address]] <return> to load and run the linker. If no arguments are given, programs are automatically linked in such a way that they will run under the control of the Shell program. In this case all program names must end in ".sh". The -s option indicates that you want the programs to run independantly of the Shell program. If no address is specified, the programs will be linked starting at the start of basic so that they may be LOADed and RUN. If an address is specified, in either hex or decimal (identify hex numbers with a preceding $ symbol), the programs will be linked from that address on. Object files will be taken from the work disk, and the library files will be looked for on the system disk.

You will then be greeted by a ">" linker prompt. Type in the names of the object files (i.e. ".o" files) you wish to link together. When you have finished this, insert the library disk and press <ARROW UP> <RETURN>. This will automatically link in the necessary functions from the standard library and system library.

When the above has been completed, hit <return>. If you get an "unresolved external reference" message then you forgot to link in something. No problem ... just link in that something now.

Once everything is linked you are asked for the name of the program to be saved on disk. If you did not specify a starting address then this program may simply be LOADed and RUN. Otherwise you must LOAD with the ",1" option and SYS to the starting address.

You may exit the linker at any time by entering x <return> at any linker prompt (>).

NOTE 1: The linker will only accept files whose names end with ".o", ".obj" or ".l".

NOTE 2: Typically, the linking commands would appear thusly:

```
>filename1.o    (link first object file)
>filename2.o    (link second object file,  if you have one)
>↑              (link libraries)
><RETURN>       (all finished)
```


PROGRAM ANOMALIES

The C PRIMER PLUS book pages listed below contain example programs that will not likely run when compiled by C POWER. Some suggestions that may make them work follow:

pp. 89, 108:
The %e conversion specifier is not supported by the library function printf(). Use %f instead.

pp. 192, 195, 345:
The tab character is not recognized by the compiler; delete references to this character. Example: change the second line on page 192 to read

```
if (ch != ' ' && ch != '/n')
```

The slash character (/) is used above to represent a back-slash.

pp. 278, 316, 321:
If the address of a scalar variable (i.e. not an array, struct, or union) of type int, unsigned, char, or pointer and with an implicit storage class of auto is passed to a function, incorrect code may be generated due to an anomaly in the compiler. Solution: explicitly declare the variable to be auto or static. For example, change the declaration on page 278 to read

```
static int x = 5, y = 10;
```

This anomaly does not appear when addresses are passed to library functions such as scanf().

pp. 422, 428, 430:
Struct variables may not be initialized. Instead, use assignments or read the data from a file if there are many values.

pp. 502-503:
Bit fields are not implemented. Use the bit operators instead.

Chapter 15:
Most of the functions described in this chapter are supported by the C POWER library though some may have a slightly different form. Consult the library documentation before using any of these functions to ensure that they are used correctly.

Appendices H (IBM PC Music) and I (C Augmentations) are not applicable.

LIBRARY INTRODUCTION

The following pages describe the functions available in the library provided with the compiler. Most of the functions are in the library file stdlib.l; the remainder are in the file math.l. See the section on the linker for information on how to link these libraries into your C program.

Each page is divided into three or four sections:

NAME

The names of the functions described on the page are listed here with a very brief description.

SYNOPSIS

The purpose of this section is to show the number, order, and types of arguments each function takes, as well as the type the function returns. If no types are specified, int is to be assumed. For example,

```
float atof(fptr)
char *fptr;
```

means that the function atof takes one argument of type pointer to char, and returns a float.

```
abs(i)
```

This means abs takes one argument of type int, and returns an int.

If a function returns a type other than int, it should be declared in the file which calls the function. For example, before any calls to malloc the following declaration should appear:

```
char *malloc();
```

Header files which are useful when using certain functions are also listed in this section. If the line

```
#include <stdio.h>
```

is listed, for example, the same line should be put into the source file which calls the function(s) described on that page.

DESCRIPTION

This section describes what the functions do, what the arguments are, and what the functions return (if anything). If a function checks the arguments for validity or does any other kind of error checking it will be described here. Otherwise it should be assumed that no error checking is done.

EXAMPLES

For some of the more complicated functions examples illustrating their use are listed. In the examples the symbols (* and *) mean left and right brace brackets, and /* ... */ means the preceeding and following code may be separated by some arbitrary amount of code.

FUNCTION INDEX

For	See	Page#
abort	exit	15
abs		14
acos	sin	35
asin	sin	35
atan	sin	35
atan2	sin	35
atof	atoi	14
atoi		14
bcmp		15
bcopy	bcmp	15
bzero	bcmp	15
cabs	hypot	22
calloc	malloc	24
ceil	floor	17
close	open	25
closedir	opendir	26
cos	sin	35
cosh	sinh	36
device		15
exit		16
exp		16
fabs	abs	14
fclose	fopen	18
feof	ferror	17
ferror		17
ffs	bcmp	15
fgetc	getc	20
fgets	gets	21
floor		17
fopen		18
fprintf	printf	27
fputc	putc	29
fputs	puts	29
fread		19
free	malloc	24
freopen	fopen	18
frexp		20
fscanf	scanf	32
fwrite	fread	19
getc		20
getchar	getc	20
gets		21
getw	getc	20
highmem		22
hypot		22
index	strcat	37
isalnum	isalpha	23
isalpha		23
isascii	isalpha	23
iscntrl	isalpha	23
isdigit	isalpha	23
islower	isalpha	23
isprint	isalpha	23
ispunct	isalpha	23
isspace	isalpha	23
isupper	isalpha	23
kernal	highmem	22
ldexp	frexp	20

For	See	Page#
log	exp	16
log10	exp	16
longjmp	setjmp	34
malloc		24
modf	floor	17
open		25
opendir		26
pow	exp	16
printf		27
putc		29
putchar	putc	29
puts		29
putw	putc	29
qsort		30
random		31
readdir	opendir	26
realloc	malloc	24
rewinddir	opendir	26
rindex	strcat	37
scanf		32
setjmp		34
sin		35
sinh		36
sprintf	printf	27
sqrt	exp	16
srandom	random	31
sscanf	scanf	32
strcat		37
strcmp	strcat	37
strcpy	strcat	37
strlen	strcat	37
strncat	strcat	37
strncmp	strcat	37
strncpy	strcat	37
sys		38
tan	sin	35
tanh	sinh	36

NAME

abs, fabs - absolute value

SYNOPSIS

```
abs(i)
int i;

float fabs(f)
float f;
```

DESCRIPTION

Abs and fabs return the absolute value of their arguments.

NAME

atoi, atof - convert strings to numbers

SYNOPSIS

```
atoi(iptr)
char *iptr;

float atof(fptr)
char *fptr;
```

DESCRIPTION

Atoi converts the string pointed to by its argument into an integer.

Atof converts the string pointed to by its argument into a float quantity.

Both functions ignore leading spaces.

EXAMPLES

```
char *s;
int i;
float pi, atof();

s = " 123";
i = atoi(s);

s = "3.14159";
pi = atof(s);
```

NAME

bcmp, bcopy, bzero, ffs - bit and byte string functions

SYNOPSIS

```
bcmp(p1, p2, len)
char *p1, *p2;

bcopy(p1, p2, len)
char *p1, *p2;

bzero(p, len)
char *p;

ffs(i)
```

DESCRIPTION

Bcmp compares len bytes of the strings p1 and p2 and returns zero if they are same, non-zero otherwise.

Bcopy copies len bytes from string p1 to string p2.

Bzero fills string p with len zeros.

Ffs returns the position of the first set bit in its argument. Bits are numbered starting at one. If the argument is zero ffs returns -1.

NAME

device - set default disk device

SYNOPSIS

```
device(n)
```

DESCRIPTION

Device sets the disk device number to be used by subsequent calls to fopen. If there are no calls to device, the default is device number 8.

EXAMPLE

```
/* open a file for reading on device 9, drive 1 */

device(9);
f = fopen("1:datafile", "r");
```

NAME

exit, abort - terminate execution

SYNOPSIS

```
exit()

abort()
```

DESCRIPTION

Exit and abort end program execution. All files opened by fopen are closed.

NAME

exp, log, log10, pow, sqrt - assorted math functions

SYNOPSIS

```
#include <math.h>

float exp(x)
float x;

float log(x)
float x;

float log10(x)
float x;

float pow(x, y)
float x, y;

float sqrt(x)
float x;
```

DESCRIPTION

Exp returns e**x.

Log returns the natural logarithm of x.

Log10 returns the base 10 logarithm of x.

Pow returns x**y.

Sqrt returns the square root of x.

NAME

ferror, feof - check for error or end of file

SYNOPSIS

```
#include <stdio.h>

ferror()

feof(stream)
FILE stream;
```

DESCRIPTION

Ferror returns non-zero if an error occured during the last disk operation, zero otherwise.

Feof returns non-zero if the specified stream has reached end of file, zero otherwise.

NAME

floor, ceil, modf - get integer part of float

SYNOPSIS

```
#include <math.h>

float floor(x)
float x;

float ceil(x)
float x;

float modf(x, ptr)
float x, *ptr;
```

DESCRIPTION

Floor returns the greatest integer not greater than x.

Ceil returns the least integer not less than x.

Modf returns the positive fractional part of x and stores the integer part indirectly through ptr.

NAME

fopen, freopen, fclose - open disk file for I/O

SYNOPSIS

```
#include <stdio.h>

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

FILE freopen(filename, mode, stream)
char *filename, *mode;
FILE stream;

fclose(stream)
FILE stream;
```

DESCRIPTION

Fopen opens a disk file for reading or writing. The string filename contains the name of the file. The first character of the string mode specifies read or write ('r' or 'w'). The default file type is sequential, but program file types may be selected (see example). Fopen returns a file number (hereafter refered to as a stream) which may be used in later I/O, or it returns zero if the file cannot be opened.

Freopen opens a file much the same as fopen does. The file stream is first closed, then if the open is successful the old stream is assigned to the new file. This is useful to assign the constant streams stdin and stdout to disk files.

Ferror should be checked after every fopen.

Fclose closes the specified file.

EXAMPLES

```
#include <stdio.h>

FILE f;

/* open sequential file for reading */

f = fopen("abc", "r");

/* open and replace program file */

f = fopen("@0:xyz,p", "w");

/* assign standard output to a disk file */

f = freopen("outfile", "w", stdout);
```

NAME

fread, fwrite - array input/output

SYNOPSIS

```
#include <stdio.h>

fread(ptr, elsize, nelem, stream)
char *ptr;
FILE stream;

fwrite(ptr, elsize, nelem, stream)
char *ptr;
FILE stream;
```

DESCRIPTION

Fread/fwrite reads/writes an array containing nelem elements each of size elsize bytes beginning at ptr from/to the specified stream.

Fread returns zero upon end of file.

EXAMPLE

```
#include <stdio.h>

#define N 500

float x[N];
FILE f;

f = fopen("datafile", "r");
fread(x, sizeof(float), N, f);
```

NAME

frexp, ldexp - split float into mantissa and exponent

SYNOPSIS

```
float frexp(value, ptr)
float value;
int *ptr;

float ldexp(value, exp)
float value;
```

DESCRIPTION

Frexp splits value into a mantissa m of magnitude less than 1 (which is returned) and an exponent exp (which is stored indirectly through ptr) such that value = m * 2**exp.

Ldexp returns value * 2**exp.

NAME

getc, getchar, fgetc, getw - input character or integer

SYNOPSIS

```
#include <stdio.h>

int getc(stream)
FILE stream;

int getchar()

int fgetc(stream)
FILE stream;

int getw(stream)
FILE stream;
```

DESCRIPTION

Getc and fgetc read a character from the specified stream.

Getchar reads a character from the standard input.

Getw reads an integer (two bytes) from the specified stream.

All of these functions return EOF upon end of file. However, since EOF is a valid integer, feof should be used to check for end of file after getw.

NAME

gets, fgets - input a string

SYNOPSIS

```
#include <stdio.h>

char *gets(s)
char *s;

char *fgets(s, n, stream)
char *s;
FILE stream;
```

DESCRIPTION

Gets inputs a string from the standard input. It reads characters into s until a newline character is encountered. The newline is replaced with a zero.

Fgets inputs a string from the specified stream. It reads n-1 characters or until a newline is encountered, whichever comes first. The newline is not replaced, but a zero is placed after the last character read.

Both functions return s upon normal completion, or NULL upon end of file.

NAME

highmem, kernal - memory configuration

SYNOPSIS

```
highmem(address)
unsigned address;

kernal(flag)
```

DESCRIPTION

Highmem sets the highest address that a C program can use. The run time stack will not go past this address, and the memory allocation functions (malloc, calloc, realloc) will not allocate memory higher than this address. The value of the argument must be one greater than the desired address. If highmem is not called, address defaults to 0xd000, which means the highest address which can be used is 0xcfff.

If the value of flag is zero, kernal configures the memory space to all RAM and disables IRQ interrupts. Otherwise kernal restores the kernal ROM and I/O space (0xd000 to 0xffff) and enables IRQ interrupts. The default configuration leaves the kernal and I/O space in place. The BASIC ROM is always turned off (except when performing floating point operations).

EXAMPLE

```
/* let program use all available memory */

kernal(0);
highmem(0xffff);
```

NAME

hypot, cabs - calculate hypotenuse

SYNOPSIS

```
#include <math.h>

float hypot(x, y)
float x, y;

float cabs(c)
struct (* float x, y; *) *c;
```

DESCRIPTION

Hypot and cabs return sqrt(x*x + y*y).

NAME

isalpha, ... - classify characters

SYNOPSIS

```
isalpha(c)

...
```

DESCRIPTION

The following functions return non-zero integers if the stated condition is true, zero otherwise.

isalpha	c is a letter
isupper	c is an upper case letter
islower	c is a lower case letter
isdigit	c is a digit
isalnum	c is a letter or digit
isspace	c is a space or newline
ispunct	c is a punctuation character
isprint	c is a printable character
iscntrl	c is a control character
isascii	c has value less than 0200

NAME

malloc, calloc, realloc, free - memory allocation

SYNOPSIS

```
char *malloc(size)
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;

char *realloc(ptr, size)
char *ptr;
unsigned size;

free(ptr)
char *ptr;
```

DESCRIPTION

Malloc returns a pointer to a block of memory containing at least size bytes.

Calloc returns a pointer to a block of zero-filled memory containing at least nelem * elsize bytes.

Realloc copies the block pointed to by ptr into a new block containing at least size bytes. Ptr must point to a block allocated by malloc, calloc, or realloc.

Free releases the block pointed to by ptr into the free memory list.

Malloc, calloc, and realloc all return the null pointer (0) if there is not enough free memory to satisfy the request.

EXAMPLE

```
/* Run time array allocation */

#define NELEM 100

char *malloc;
int *t;

t = (int *) malloc(NELEM * sizeof(int));

/* ... */

free(t); /* done with array */
```

NAME

open, close - BASIC style open

SYNOPSIS

```
open(fileno, device, secaddr, name)
char *name;

close(fileno);
```

DESCRIPTION

The arguments of open correspond exactly to the file number, device number, secondary address, and file name arguments of the BASIC OPEN command. Consult a Commodore 64 manual for the meanings of the arguments. Similarly, close corresponds to the BASIC CLOSE command. Open returns zero if the file can't be opened, non-zero otherwise. As with fopen, ferror should be checked after opening a write file.

The file number argument may be used any place a stream (i.e. a value returned by fopen) is used (see example). File numbers 1 through 4 are reserved for system use. If open and fopen are to be used at the same time, file numbers passed to open should be limited to the range 5 through 9.

EXAMPLE

```
/* display disk file on screen */

#include <stdio.h>

char c;

open(5, 8, 5, "filename,s,r");

for ((c = getc(5)) != EOF)
    putchar(c);

close(5);
```

NAME

opendir, readdir, rewinddir, closedir - directory functions

SYNOPSIS

```
#include <dir.h>

opendir()

struct direct *readdir()

rewinddir()

closedir()
```

DESCRIPTION

Opendir opens a disk directory for reading. If the directory can't be opened NULL is returned.

Readdir reads the next directory entry and returns a pointer to it. If there are no more entries NULL is returned. See the header file dir.h and the VIC-1541 User's Manual page 56 for the format of a directory entry.

Rewinddir causes readdir to read the first entry upon the next call.

Closedir closes the directory for.

EXAMPLE

```
/* Display contents of disk directory */

#include <dir.h>
#include <stdio.h>

struct direct *dp;

opendir();
for (dp = readdir(); dp != NULL; dp = readdir())
    puts (dp->name);
closedir();
```

NAME

printf, fprintf, sprintf - formatted output

SYNOPSIS

```
#include <stdio.h>

printf(control [, arg] ...)
char *control;

fprintf(stream, control [, arg] ...)
FILE stream;
char *control;

sprintf(s, control [, arg] ...)
char *s, *control;
```

DESCRIPTION

These functions output optional lists of arguments according to a format specified in the null terminated control string. Printf sends output to the standard output. Fprintf sends output to the specified stream. Sprintf places output in the string s. Sprintf also places a null character in s after the last output character.

The control string may contain ordinary characters, which are output, and conversion specifiers, which specify how an argument is to be formatted. Each conversion specifier begins with a percent character (%) and is followed by:

An optional dash (-) which indicates left adjustment of the argument in the output field. Right adjustment is the default.

An optional number indicating the minimum field width. A converted argument will not be truncated even if it won't fit in the specified field. If the first digit of the field width is zero, the field will be padded with zeros; otherwise it will be padded with spaces. The maximum field width is 128 characters.

An optional period (.) followed by a number indicating the precision for a float or string argument. For floats the precision indicates the number of digits to be printed after the decimal point (default is six). If the precision is explicitly zero no decimal point is printed. For strings the precision indicates the maximum number of characters from the string to be printed (default is the whole string).

A letter indicating the type of conversion to be performed. The following letters are recognized:

d - an integer argument is printed as a possibly signed decimal number

u - an integer argument is printed as an unsigned decimal number

o - an integer argument is printed as an octal number

x - an integer argument is printed as a hexadecimal number

f - a float argument is printed

s - a character pointer argument is assumed to point to a null terminated string which is printed

c - an integer argument is assumed to be a character and is printed as such

For each conversion specifier a corresponding argument of an appropriate type must be provided.

To output a percent character use %%.

A star (*) may be used in place of the field width or precision. The value will be taken from an integer argument.

EXAMPLES

```
printf("%d %f", 123, 3.14);

/* output: 123 3.140000 */

printf("%05x", 0x2a3);

/* output: 002a3 */

printf ("abc%-*.*fxyz", 9, 2, 12.3456);

/* output: abc12.34 xyz */
```

NAME

putc, putchar, fpuc, putw - output a character or integer

```
#include <stdio.h>

putc(c, stream)
FILE stream;

putchar(c)

fputc(c, stream)
FILE stream;

putw(i, stream)
FILE stream;
```

DESCRIPTION

Putc and fputc write the character c to the specified stream.

Putchar writes the character c to the standard output.

Puts writes the integer i (two bytes) to the specified stream.

NAME

puts, fputs - output a string

SYNOPSIS

```
#include <stdio.h>

puts(s)
     char *s;

fputs(s, stream)
char *s;
FILE stream;
```

DESCRIPTION

Puts writes the null terminated string s to the standard output. Puts also writes a newline character after the string.

Fputs writes the null terminated string s to the specified stream. Fputs does not write an additional newline character.

NAME

qsort - general purpose sort

SYNOPSIS

```
qsort(base, nel, elsize, comp)
char *base;
int (*comp)();
```

DESCRIPTION

Qsort sorts an array beginning at base containing nel elements each of size elsize. Comp points to a function which compares elements. The function must take two pointers to elements and return an integer less than, equal to, or greater than zero as the first element is less than, equal to, or greater than the second.

EXAMPLE

```
/* Sort array of floats */

#define NELEM 100

float t[NELEM];
int fcomp();

qsort(t, NELEM, sizeof(float), fcomp);

/* ... */

fcomp(p1, p2)
float *p1, *p2;
(*
        if (*p1 < *p2)
            return (-1);
        else if (*p1 == *p2)
            return (0);
         else
            return (1);
*)
```


NAME

random, srandom - random number generator

SYNOPSIS

```
random()

srandom(seed);
```

DESCRIPTION

Random returns a pseudo-random integer.

Srandom sets the state of the random number generator. If srandom is called twice with the same seed the same sequence of random integers will be generated.

NAME

scanf, fscanf, sscanf - formatted input

SYNOPSIS

```
#include <stdio.h>

scanf(control [, arg] ...)
char *control;

fscanf(stream, control [, arg] ...)
FILE stream;
char *control;

sscanf(s, control [, arg] ...)
char *s, *control;
```

DESCRIPTION

These functions read sequences of characters, perform conversions specified by the control string on them, and store the converted values indirectly through pointer arguments. Scanf reads from the standard input, fscanf reads from the specified stream, and sscanf reads from the string s.

The control string may contain blanks and newlines, which may match optional blanks and newlines from the input, other ordinary characters, which must match corresponding characters from the input, and conversion specifiers. Each conversion specifier begins with a percent character (%) and is followed by:

An optional star (*) which suppresses assignment of the converted value.

An optional number which specifies the maximum field width. Characters are read up to the first unrecognized character for the type of conversion being performed or until the number of characters read equals the field width, whichever comes first. If no field width is specified characters are read up to the first unrecognized character.

A letter indicating the type of conversion to be performed on the field. The following letters are recognized:

d- the field is expected to contain a possibly signed decimal number which is converted into an integer

x- the field is expected to contain a hexadecimal number which is converted into an integer

o- the field is expected to contain an octal number which is converted into an integer

f- the field is expected to contain a possibly signed decimal number with an optional decimal point and exponent which is converted into a float

s- no conversion is performed - the field is copied into a string argument with a null character appended

c- the field contains a single character which is copied into a character argument

For each conversion specifier (except for those which suppress assignment) there must be a corresponding argument which is a pointer to an appropriate type. For example, d conversion requires that there be a pointer to an int or an unsigned.

To match a percent character from the input use %%.

These functions return EOF upon end of file; otherwise they returned the number of conversions successfully performed. This number may be less than the number of conversion specifiers if, for example, characters in the control string do not match corresponding characters from the input.

To input strings with embedded spaces use gets or fgets.

EXAMPLES

```
#include <stdio.h>

int i;
float f;
char s[50], c;

scanf("%d %f", &i, &f);

/*
     input: 123 456
     result: i = 123, f = 456.0
*/

scanf("%3d %5f", &i, &f);

/*
     input: 436504.3683
     result: i = 436, f = 504.3
*/

scanf("%11s Spain %c", s, &c);

/*
     input: The rain in Spain falls mainly ...
     result: s = "The rain in", c = 'f'
*/
```

NAME

setjmp, longjmp - long range goto

SYNOPSIS

```
#include <setjmp.h>

setjmp(env)
jmp buf env;

longjmp(env, val)
jmp buf env;
```

DESCRIPTION

Setjmp stores its stack environment in env and returns zero.

Longjmp restores the stack environment saved by setjmp and returns in such a way that it appears that the original call to setjmp has returned with the value val.

The calls to setjmp and longjmp may be in different functions, but the function containing the setjmp call must not have returned before the call to longjmp.

EXAMPLE

```
#include <setjmp.h>

int errno, error;
jmp buf env;

errno = setjmp(env);
if (errno != 0) (*
    printf ("error #%d", errno);
    exit();
*)

/* ... */

if (error)
    longjmp(env, 1);
```

NAME

sin, cos, tan, asin, acos, atan, atan2 - trig functions

SYNOPSIS

```
#include <math.h>

float sin(x)
float x;

float cos(x)
float x;

float tan(x)
float x;

float asin(x)
float x;

float acos(x)
float x;

float atan(x)
float x;

float atan2(x, y)
float x, y;
```

DESCRIPTION

Sin, cos, and tan return the sine, cosine, and tangent of x respectively. X is measured in radians.

Asin, acos, and atan return the arcsine, arccos, and arctangent of x respectively.

Atan2 returns the arctangent of x/y.

NAME

sinh, cosh, tanh - hyperbolic functions

SYNOPSIS

```
#include <math.h>

float sinh(x)
      float x;

float cosh(x)
float x;

float tanh(x)
float x;
```

DESCRIPTION

Sinh, cosh, and tanh return the hyperbolic sine, cosine, and tangent of x respectively.

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, index, rindex - string functions

SYNOPSIS

```
#include <strings.h>

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

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

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

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

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

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

strlen(s)
char *s;

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

char *rindex(s, c)
char *s, c;
```

DESCRIPTION

All of the following functions operate on character strings terminated with zero.

Strcat and strncat concatenate the strings s1 and s2 and leave the result in s1. Strncat copies at most n characters from s2. Both functions return s1.

Strcmp and strncmp compare the strings s1 and s2 and return an integer less than, equal to, or greater than zero as s1 is lexically less than, equal to, or greater than s2. Strncmp compares at most n characters.

Strcpy and strncpy copy s2 into s1. Strncpy copies at most n characters. Both functions return s1.

Strlen returns the number of non-zero characters in the string s.

Index/rindex returns a pointer to the leftmost/rightmost occurence of the character c in the string s. If the character is not in the string a null pointer (0) is returned.

NAME

sys - call a machine language subroutine

SYNOPSIS

```
sys(address, a, x, y)
unsigned address;
char *a, *x, *y;
```

DESCRIPTION

Sys loads the accumulator, x, and y registers of the 6510 processor with the values pointed to by a, x, and y respectively then jumps to the subroutine located at the specified address. Upon completion of the subroutine the (possibly) new values contained in the registers are stored indirectly through the pointer arguments. Sys returns zero if the carry flag is clear after the subroutine call; otherwise it returns one.

This function allows the programmer to combine assembler and C code in one program without having to use a special assembler. Another use of sys is to access kernal routines not otherwise supported by the standard library.

EXAMPLE

```
char *s, x, y;

s = "this string will be written to the screen";

while (*s++)
     sys(0xffd2, s, &x, &y);

/* ffd2 is the kernal routine for printing a character */
```

Warranty Policies

Part of the C POWER disk is protected against copying, so there are some Pro-Line policies that you should be made aware of.

1. Original disks are covered under warranty to the registered original purchaser for a period of one year from date of purchase, and if determined not to have failed through mishandling or misuse, will be exchanged directly by Pro-Line for a fee of $5.00 postage and handling.

2. Registered original purchasers may, while the program remains in production, exchange their original C-POWER disk for an UP-DATED version (should such be released and become available) at a cost of $5.00 including postage and handling.

3. A BACK-UP copy of C POWER is available to registered original purchasers directly from Pro-Line Software Ltd. at a cost of $20.00 including postage and handling.

Liability Disclaimer

All Pro-Line Software Ltd. computer programs, whether supplied on magnetic media such as but not limited to cassette tapes or diskettes or in any other form, any associated devices and any instructions or manuals, are sold on an "as is" basis without warranty of any kind expressed or implied except as above stated. Pro-Line Software Ltd. does not warrant the fitness of any program for any purpose nor does Pro-line Software Ltd. warrant the accuracy, quality or freedom from errors of any programs, devices, instructions or manuals except as above stated.

Pro-Line Software Ltd., their distributors, agents and retailers can assume no responsibility for any consequential, incidental or other liability arising from the use, the inability to use or the attempted use of any program nor any lost profit or any lost savings however incurred.

* * * * * * *

NOTE: As Pro-Line Software is constantly striving to improve it's software programs, all specifications are subject to change without notice.

* * * * * * *

C SHELL UTILITIES [1]

Name

find - pattern matcher

Synopsis

find [-x] [-n] [-f] pattern [filename] [filename] ...

Description

Find searches the input and writes all lines matching the pattern to the standard output. Input is taken from the named files, if any, otherwise it is taken from the standard input.

The options are:

-x : Only write lines which don't match the pattern

-n : Write the line number of each matched line.

-f : Write the file name before each matched line.

Patterns may consist of ordinary characters which match corresponding characters in the input, and special characters or meta characters which match special patterns.

These characters are:

? Match any character.

* Match zero or more occurences of the previous element of the pattern.

% Match the start of a line.

$ Match the end of a line.

[class] Match any character belonging to the specified character class. A character class is simply a list of characters. For example, [aeiou] matches any lower case vowel. Character classes may be abbreviated if they contain sequences of consecutive letters or digits. For example, [A-Z] matches any upper case letter, and [aeiou0-9] matches any vowel or digit.

[!class] Match any character NOT belonging to the character class.

Preceding a metacharacter with an "at" symbol (@) will cause it to be treated as an ordinary character. Thus @? matches a single question mark, and @@ matches a single "at" symbol.

[1] Adapted from programs in "Software Tools" by Kernihan & Ritchie

Examples

abc

Match lines containing the string "abc".

%abc

Match lines starting with the string "abc".

x?y

Match lines containing an x, followed by any character, followed by a y.

x?*y$

Match lines containing and x and ending with a y.

%[a-z]*[A-Z][A-Z]*$

Match lines starting with a possibly empty string of lower case letters and ending with a non-empty string of upper case letters.

[_a-zA-Z][_a-zA-Z0-9]*[ ]*=[ ]*[0-9][0-9]*

Match lines containing an assignment of an integer constant to a C identifier. Note that this pattern would have to be enclosed in double quotes since it contains spaces.

Name

format - text formatter

Synopsis

$ format [filename] [filename] ...

Description

Format reads text from the standard input if no arguments are given, otherwise it reads from the specified files. Output is written to standard output.

Input may consist of ordinary text, which is filled and justified by default, and formatting commands. Formatting commands consist of a period (.) in the first character position of a line, a two character code, and for most commands an optional argument. The commands recognized by the formatter are:

.bp n

Begin page numbered n. This forces the start of a new page with page number n. The default for n is the current page number plus 1.

.br

Cause a break. This forces any accumulated text not yet written to be written immediately. Several commands implicitly cause a break before they perform their function. These are: .bp, .ce, .fi, .ne, .nf, .sp, and .ti.

.ce n

Center the next n lines. Default: n=1

.fi

Fill text. Text will be filled (output lines will contain as many words as possible) and right justified (right margins will be lined up). Format fills by default.

.fo /left footer/center footer/right footer/

Set footer (bottom of page titles). The strings "left footer", "center footer", "right footer" will be written at the bottom of each page left justified, centered and right justified respectively. All occurences of the character '#' in the strings will be replaced with the current page number.

.he /left header/center header/right header/

Set header (top of page titles).

.in n

Set indentation. N spaces will be placed at the start of each output line. Default: n=0

.ls n

Set line spacing. N-1 blank lines will be inserted between each line of text. Default: n=1

.m1 n

Set margin above and including the header to n, Default: n=3

.m2 n

Set margin below header to n. Default: n=3

.m3 n

Set margin above the footer to n. Default: n=3

.m4 n

Set margin below and including the footer to n. Default: n=3

.ne n

Need n lines. If there are fewer that n lines remaining on the current page, then skip to a new page.

.nf

Stop filling text. Lines will be copied from input to output without change except for indentation and line spacing.

.pl n

Set page length (number of lines per page). Default: n=66

.rm n

Set right margin (the rightmost character position to be written to). Default: n=60

.sp n

Write n blank lines.

.ti n

Temporary indent. The next output line (and only that line) will be given an indentation of n rather than the value set by .in. Default: n=0

Numeric arguments may be specified in two ways: as absolute (unsigned) integers, or as signed integers. Absolute arguments are assigned to parameters in the obvious way:

```
.ls 2
```

sets the line spacing to 2. Signed arguments indicate a change in the current value of the parameter being set; the value of the argument is added to or subtracted from the current value. For example, the commands

```
.pl 66
.pl -10
```

will set the page length to 56, and

```
.in 10
.in +5
```

will cause a temporary indent of 15.

Blank lines and lines starting with spaces occuring in the input are special cases. Blank lines cause a break and a number of blank lines equal to the current line spacing to be written. Lines starting with spaces cause a break and a temporary indent of +n where n is the number of spaces before the first non-space character on the line.

Name

print - page files

Synopsis

$ print [filename] [filename] ...

Description

Print writes the named files (or the standard input if none are specified) to the standard output with margins at the top and bottom of each page, and a header at the top of each page.

Name

trim - code optimizer

Synopsis

$ trim [filename.o] ...

Description

Trim optimizes the .o (or .obj) files produced by the compiler prior to LINKing. You can expect to reduce compiled file sizes in the order of 8 to 10 percent.