VAX C
Language Summary

digital
software

VAX C Language Summary

Order Number: AA–P788B–TE

April 1985

This document contains a syntax summary of the VAX C language constructs, the VAX C Run-Time Library functions and macros, and the VAX/VMS commands used to compile and link a VAX C program.

Revision/Update Information: This revised document supersedes the *VAX–11 C Language Summary* (Order No. AA–P788A–TE).

Operating System and Version: VAX/VMS Version 4.0 or later.

Software Version: VAX C Version 2.0.

digital equipment corporation
maynard, massachusetts

Revised, April 1985

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

ZK–2755

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by TEX, the typesetting system developed by Donald E. Knuth at Stanford University. TEX is a registered trademark of the American Mathematical Society.

Contents

Preface

Conventions Used in This Document

Convention	Meaning
char	Language keywords, and VAX C Run-Time Library functions and macros are represented in bold type.
file_spec, stdio	Arguments to functions or macros, and .H library module names are represented in italics.
$RUN CPROG RETURN	In interactive examples, the user's response to a prompt is printed in red; system prompts are printed in black.

```
globaldef
float x;
.
.
.
x = 5;
```

VAX C specific constructs are represented in blue.

A vertical ellipsis indicates that not all of the text of a program or program output is illustrated. Only relevant material is shown in the example.

```
option, . . .
```

A horizontal ellipsis indicates that additional parameters, options, or values can be entered. A comma that precedes the ellipsis indicates that successive items must be separated by commas.

Convention	Meaning
sc-specifier ::= **auto** **static** **extern** **register**	In syntax definitions, items appearing on separate lines are are mutually exclusive alternatives.
[a|b]	Braces surrounding two or more items separated by a vertical bar (|) indicate a choice; you must choose one of the two syntactic elements.

Associated Documents

- *Programming in VAX C*—For a complete guide to program development on VAX/VMS using the VAX C programming language
- *Installing VAX C*—For a guide to installing the VAX C software
- *VAX/VMS Master Index*—For a guide to working with the VAX machine architecture or the VAX/VMS system services

  This index lists manuals which cover the individual topics concerning access to VAX/VMS.

Chapter 1

The DCL Command Line

This chapter describes the CC and LINK commands of the DIGITAL Command Language (DCL) and the qualifiers used with both commands.

1.1 The CC Command

The DIGITAL Command Language (DCL) command, CC, compiles one or more VAX C source files into one or more object files. The source file or files compiled into an object module is called the *compilation unit*.

Syntax:

```
CC[/qualifier . . . ] file-spec-list
```

File Specification Syntax:

```
file-spec[/qualifier . . . ]
file-spec-list, file-spec[/qualifier . . . ]
file-spec-list + file-spec[/qualifier . . . ]
```

Command Qualifiers	Defaults
/[NO]CROSS_REFERENCE	/NOCROSS_REFERENCE
/[NO]DEBUG[=option]	/DEBUG=TRACEBACK
/[NO]DEFINE[=(definition list)]	/NODEFINE
/DIAGNOSTICS	/NODIAGNOSTICS
/G_FLOAT	/NOG_FLOAT
/LIBRARY	
/[NO]LIST	/NOLIST (interactive mode) /LIST (batch mode)

Command Qualifiers	Defaults
/[NO]MACHINE_CODE[=option]	/NOMACHINE_CODE
/[NO]OBJECT	/OBJECT
/[NO]OPTIMIZE[=NODISJOINT]	/OPTIMIZE
/SHOW[=(option, . . .)]	/SHOW=(NOBRIEF, NODICTIONARY, NOEXPANSION, NOINCLUDE, NOINTERMEDIATE, NOSTATISTICS, NOSYMBOLS, NOTRANSLATION, SOURCE, TERMINAL)
/STANDARD=[NO]PORTABLE	/STANDARD=NOPORTABLE
/[NO]UNDEFINE[=(undefine list)]	/NOUNDEFINE
/[NO]WARNINGS[=(option-list)]	/WARNINGS

NOTE

The only qualifier that *must* be used with a file specification is the /LIBRARY qualifier. You cannot place this qualifier on the CC command.

1.2 The LINK Command

The DCL LINK command combines one or more object modules into one image file. If your program contains references to the VAX C Run-Time Library (RTL) functions, read "Specifying Libraries to the Linker" in this section before you link your programs.

Syntax:

```
LINK[/qualifier . . . ]  file-spec[/qualifier . . . ], . . .
```

Command Qualifiers	Defaults
/BRIEF	None
/[NO]CONTIGUOUS	/NOCONTIGUOUS
/[NO]CROSS_REFERENCE	/NOCROSS_REFERENCE
/[NO]DEBUG[=file_spec]	/NODEBUG
/[NO]EXECUTABLE[=file_spec]	/EXECUTABLE
/FULL	None
/HEADER	None
/[NO]MAP[=file_spec]	/NOMAP
/POIMAGE	None
/PROTECT	None
/[NO]SHAREABLE[=file_spec]	/NOSHAREABLE
/[NO]SYMBOL_TABLE[=file_spec]	/NOSYMBOL_TABLE
/[NO]SYSLIB	/SYSLIB
/[NO]SYSSHR	/SYSSHR
/[NO]SYSTEM[=base_address]	/NOSYSTEM
/[NO]TRACEBACK	/TRACEBACK
/[NO]USERLIBRARY[=table[, . . .]]	None

/INCLUDE=(module_name[, . . .])	None
/LIBRARY	None
/OPTIONS	None
/SELECTIVE_SEARCH	None

NOTE

The only qualifiers that *must* be used with a file specification are the /INCLUDE, /LIBRARY, /OPTIONS, and /SELECTIVE_SEARCH qualifiers. You cannot place these qualifiers on the LINK command.

Specifying Libraries to the Linker:

If you use any of the VAX C Run-Time Library functions, you must specify libraries for the linker to search in order to resolve references to the functions. You can specify these libraries using the /LIBRARY qualifier on the LINK command line, or you can define the logical name LNK$LIBRARY_n to be the name of an object library. You must define the logicals LNK$LIBRARY_n as the libraries SYS$LIBRARY:VAXCCURSE.OLB, SYS$LIBRARY:VAXCRTLG.OLB, and SYS$LIBRARY:VAXCRTL.OLB. Depending on the needs of your program, you may have to access one, two, or all three libraries. In any case, you must adhere to the following rules for defining libraries for the linker to search:

1. If you do not need to use the Curses Screen Management package of VAX C RTL functions and macros, and you do not use the /G_FLOAT qualifier on the CC command line, you must define the logical as follows:

   ```
   $ DEFINE  LNK$LIBRARY  SYS$LIBRARY:VAXCRTL.OLB RETURN
   ```

2. If you plan to use the /G_FLOAT qualifier with the CC command line, but do not plan on using Curses, you must define the logicals as follows:

```
$ DEFINE  LNK$LIBRARY SYS$LIBRARY:VAXCRTLG.OLB [RETURN]
$ DEFINE  LNK$LIBRARY_1 SYS$LIBRARY:VAXCRTL.OLB [RETURN]
```

3. If you plan to use the Curses Screen Management package, but do not plan to use the /G_FLOAT qualifier, you must define the logicals as follows:

```
$ DEFINE  LNK$LIBRARY SYS$LIBRARY:VAXCCURSE.OLB [RETURN]
$ DEFINE  LNK$LIBRARY_1 SYS$LIBRARY:VAXCRTL.OLB [RETURN]
```

4. Finally, if you plan to use both Curses and the /G_FLOAT qualifier, you must define the three logicals in the following order:

```
$ DEFINE  LNK$LIBRARY SYS$LIBRARY:VAXCCURSE.OLB [RETURN]
$ DEFINE  LNK$LIBRARY_1 SYS$LIBRARY:VAXCRTLG.OLB [RETURN]
$ DEFINE  LNK$LIBRARY_2 SYS$LIBRARY:VAXCRTL.OLB [RETURN]
```

The order of the specified libraries determines which versions of the VAX C RTL functions are found by the linker first. If the linker does not find the function code, or if LNK$LIBRARY_n is undefined, it assumes that the function is not a VAX C RTL function, and checks other default libraries before it assumes that the program is in error.

Instead of using the object code of the VAX C RTL functions, you can, as an option, use the VAX C RTL as a shareable image. To use the VAX C RTL as a shareable image, check with your system manager to make sure that the VAX C compiler and software were installed so as to allow access to the shared images. Specifically, check to make sure that the system manager answered YES to step 4 listed in *Installing VAX C*. If that has been done, you can create an options file.

If you do *not* use the /G_FLOAT qualifier on the CC command, create an options file, OPTIONS_FILE.OPT, containing the following line:

```
SYS$SHARE:VAXCRTL.EXE/SHARE
```

If you *do* use the /G_FLOAT qualifier on the CC command, create an options file containing the following line:

```
SYS$SHARE:VAXCRTLG.EXE/SHARE
```

You must *not* include the libraries SYS$SHARE:VAXCRTL.EXE and SYS$SHARE:VAXCRTLG.EXE in the same options file.

Once you have created the appropriate options file, OPTIONS_FILE.OPT, you can compile and link your program with the following commands:

```
$ CC  PROGRAM.C [RETURN]
$ LINK  PROGRAM.OBJ, OPTIONS_FILE/OPT [RETURN]
```

Chapter 2

VAX C Language Constructs

This chapter describes the language keywords and other constructs needed for programming in VAX C.

2.1 Data Type Keywords

Type Specifiers:

32-bit signed or unsigned:

```
int
long
long int
```

```
unsigned int
unsigned long
unsigned long int
```

16-bit signed or unsigned:

```
short
short int
unsigned short
unsigned short int
```

8-bit signed or unsigned:

```
char
unsigned char
```

F_floating format:

```
float
```

D_floating or G_floating format:

```
double
long float
```

Aggregate types:

```
struct
union
```

Enumerated type:

```
enum
```

Type of function return value:

```
void
```

Type declaration:

```
typedef
```

Storage class specifiers:

```
auto
register
static
extern
globaldef
globalref
globalvalue
```

Storage class modifiers:

```
readonly
noshare
```

2.2 Precedence of Operators

The operators are listed from highest precedence to lowest. In the binary operator category, operators appearing on higher lines within the category have a higher precedence than the other binary operators.

Category	Association	Operator
Primary	Left to right	( ) [ ] – > .
Unary	Right to left	! ~ ++ – – (*type*) – * & **sizeof**
Binary	Left to right	* / %
		+ –
		<< >>

		< <= > >=
		== !=
		&
		^
		|
		&&
		||
Conditional	Right to left	?:
Assignment	Right to left	= += -= *= /= %= > >= < <= &= ^= |=
Comma	Left to right	,

2.3 Statements

Syntax:

[*expression*] ;

identifier : *statement*

{ [*declaration-list*] [*statement-list*] }

case [*constant-expression* | default] : *statement-list*

if (*expression*) *statement* [**else** *statement*]

while (*expression*) *statement*

do *statement* **while** (*expression*)

for ([*expression*] ; [*expression*] ; [*expression*])
statement

switch (*expression*) *statement*

break ;

continue ;

return [*expression*] ;

goto *identifier* ;

entry[1]

---

1 Reserved for future use.

2.4 Conversion Rules

Arithmetic Conversion

Any operand of type:	Is converted to:
char	**int**
short	**int**
unsigned char	**unsigned int**
unsigned short	**unsigned int**
float	**double**

If operand type is:	The result and the other operands are:
double	**double**
unsigned	**unsigned**

Otherwise, both operands are:	And the result is:
int	**int**

Function Argument Conversion

Any argument of type:	Is converted to type:
float	**double**
char	**int**
short	**int**
unsigned char	**unsigned int**
unsigned short	**unsigned int**
array	pointer to array
function	pointer to function

2.5 VAX C Escape Sequences

Character	Mnemonic	Escape Sequence
newline	NL	\n
horizontal tab	HT	\t
vertical tab	VT	\v
backspace	BS	\b
carriage return	CR	\r
form feed	FF	\f
backslash	\	\\
apostrophe	'	\'
quotes	"	\"
bit pattern	ddd	\ddd

In the previous list, the characters, ddd, signify from one to three octal digits that give the value of an ASCII character. For example, \0 signifies the ASCII NUL character.

2.6 Preprocessor Control Lines

Syntax:

#define *identifier*[([*param1*, . . . *param2*])] *token-string*

#undef *identifier*

#dictionary *cdd-path*

#include <*file-spec*>
#include "*file-spec*"
#include *module-name*

#if *constant-expression*

#ifdef *identifier*

#ifndef *identifier*

```
#else
#endif

#[line]  constant  string
#[line]  constant  identifier

#module  identifier  identifier
#module  identifier  string
```

2.7 Record Management Services (RMS)

The RMS functions can be expressed in terms of the following general descriptions.

Syntax:

```
#include  rms-module
int  sys$name(pointer, [error_function],
             [success_function])

struct  rms_structure  *pointer;
int     (*error_function)(),  (*success_function)();
```

rms-module

The name of one of the modules in the following list:

Module	Description	Structure Tag
fab	File access block	FAB
rab	Record access block	RAB
nam	Name block	NAM
xaball	Allocation XAB	XABALL
xabdat	Date and time XAB	XABDAT
xabfhc	File header characteristics XAB	XABFHC
xabkey	Indexed file key XAB	XABKEY
xabpro	Protection XAB	XABPRO
xabrdt	Revision date and time XAB	XABRDT

Module	Description	Structure Tag
xabsum	Summary XAB	XABSUM
xabtrm	Terminal Control XAB	XABTRM
rmsdef	Completion status codes	—
rms	All RMS modules	All tags

sys$name

The name of the RMS function being called.

pointer

A pointer to an RMS structure which (optionally) has been initialized by the following prototypes:

Prototype	Description
cc$rms_fab	Initializes the file access block (FAB)
cc$rms_rab	Initializes the record access block (RAB)

cc$rms_nam	Initializes the name block (NAM)
cc$rms_xaball	Initializes the allocation XAB (XABALL)
cc$rms_xabdat	Initializes the date and time XAB (XABDAT)
cc$rms_xabfhc	Initializes the file header characteristics XAB (XABFHC)
cc$rms_xabkey	Initializes the indexed file key XAB (XABKEY)
cc$rms_xabpro	Initializes the protection XAB (XABPRO)
cc$rms_xabrdt	Initializes the revision date and time XAB (XABRDT)
cc$rms_xabsum	Initializes the summary XAB (XABSUM)
cc$rms_xabtrm	Initializes the terminal control XAB (XABTRM)

error_function
The name of a signal handling function to be called if an error occurs (optional).

success_function
The name of a function to be called if the RMS function is successful (optional).

rms_structure
The type of structure being pointed to by *pointer*.

The RMS functions return an integer status value.

Chapter 3

VAX C Run-Time Library Functions and Macros

This chapter describes the syntax of the VAX C Run-Time Library functions and macros.

3.1 Function and Macro Syntax Listing

abort Function

```
abort()
```

Return Values:

None.

abs Function

```
#include math

int    abs(integer)
double fabs(x)

int    integer;
double x;
```

Return Values:

Absolute value of *x*.

access Function

```
#include stdio

int  access(file_specification, mode)

char *file_specification;
int  mode;
```

Return Values:

0, if the access (privilege) is allowed; –1, if not.

Notes:

mode
Argument

0	Checks to see if the file exists
1	Execute access

2	Write access (implies delete access)
4	Read access

acos Function

```
#include math

double acos(x)

double x;
```

Return Values:

Arc cosine of *x* in the range 0 to *pi* .

addch Macro and waddch Function

```
#include curses

addch(ch)
waddch(win, ch)

WINDOW  *win;
char     ch;
```

Return Values:

OK on success; ERR, if illegal scrolling occurs.

addstr Macro and waddstr Function

```
#include curses

addstr(str)
waddstr(win, str)
```

```
WINDOW  *win;
char    *str;
```

Return Values:

OK on success; ERR, if illegal scrolling occurs.

alarm Function

```
int  alarm(seconds)
unsigned  seconds;
```

Return Values:

The number of seconds remaining from a previous alarm request.

asin Function

```
#include math

double asin(x)

double x;
```

Return Values:

Arc sine of x in the range $-pi/2$ to $pi/2$. When $|x| > 1$, this function returns 0 and sets the variable, errno, to EDOM.

atan Function

```
#include math

double atan(x)

double x;
```

Return Values:

Arc tangent of x in the range $-pi/2$ to $pi/2$.

atan2 Function

```
#include math
double  atan2(x, y)
double  x, y;
```

Return Values:

Arc tangent of x/y in the range $-pi$ to pi .

atof, atoi, and atol Functions

```
#include math

double atof(nptr)
int    atoi(nptr)
long   atol(nptr)

char   *nptr;
```

Return Values:

Numeric value of the string as a **double** (**atof**), as an **int** (**atoi**), and as a **long int** (**atol**).

Notes:

In VAX C, **atol** is functionally equivalent to **atoi**.

atoi Function

See **atof**.

atol Function

See **atof**.

box Function

```
#include curses

box(win, vert, hor)

WINDOW  *win;
char    vert, hor;
```

Return Values:

OK on success; ERR on failure.

brk and sbrk Functions

```
char  *brk(addr)
char  *sbrk(incr)

unsigned incr, addr;
```

Return Values:

Lowest virtual address not used by the program (**brk**) and the old break address if the new break address is successfully set (**sbrk**). If the program requests too much memory, both return –1.

cabs and hypot Functions

```
#include math

double cabs(z)
double hypot(x, y)
```

```
double x, y;
struct
   {
      double x, y;
   } z;
```

Return Values:

Square root of the sum of their two squared arguments *(***sqrt***(x*x + y*y))* .

calloc and malloc Functions

```
char *calloc(number, size)
char *malloc(size)

unsigned number, size;
```

Return Values:

Pointer to the first byte of the allocated space; 0, if the memory cannot be allocated.

ceil Function

```
#include math

double  ceil(x)

double  x;
```

Return Values:

Smallest integer that is greater than or equal to *x*.

cfree and free Functions

```
int  cfree(pointer)
int  free(pointer)

char  *pointer;
```

Return Values:

0 if the area previously allocated by **calloc**, **malloc**, or **realloc** is successfully freed; –1 if not.

Notes:

If porting programs across systems, use **free** with **malloc** and use **cfree** with **calloc**.

chdir Function

```
int   chdir(name)
char  *name;
```

Return Values:

0, if the directory is successfully changed; –1 if not.

chmod Function

```
int   chmod(name, mode)
char     *name;
unsigned mode;
```

Return Values:

0, if the change is successful; –1, if not.

Notes:

mode Argument	Privilege
0400	OWNER:READ
0200	OWNER:WRITE
0100	OWNER:EXECUTE
0040	GROUP:READ
0020	GROUP:WRITE
0010	GROUP:EXECUTE
0004	WORLD:READ
0002	WORLD:WRITE
0001	WORLD:EXECUTE

chown Function

```
int  chown(name, owner, group)

char     *name;
unsigned owner, group;
```

Return Values:

0 if the owner UIC of the file is changed; –1 if not.

clear Macro and wclear Function

```
#include curses

clear()
wclear(win)

WINDOW  *win;
```

Return Values:

OK on success; ERR on failure.

clearerr Macro

```
#include stdio
clearerr(file_ptr)
FILE *file_ptr;
```

Return Values:

None.

clearok Macro

```
#include curses

clearok(win, boolf)

WINDOW  *win;
bool    boolf;
```

Return Values:

OK on success; ERR on failure.

close Function

```
int  close(file_desc)

int  file_desc;
```

Return Values:

0, if the file is successfully closed; –1, if the file descriptor is undefined or if an error occurs while the file is being closed.

clrattr Macro and wclrattr Function

```
#include curses

clrattr(attr)
wclrattr(win, attr)

WINDOW  *win;
int     attr;
```

Return Values:

OK on success; ERR on failure.

Notes:

attr Argument

_BLINK	Clear blinking attribute
_BOLD	Clear bold face attribute
_REVERSE	Clear reverse video attribute
_UNDERLINE	Clear underline attribute

This function and macro are VAX C specific.

clrtobot Macro and wclrtobot Function

```
#include curses

clrtobot()
wclrtobot(win)

WINDOW  *win;
```

Return Values:

OK on success; ERR on failure.

clrtoeol Macro and wclrtoeol Function

```
#include curses

clrtoeol()
wclrtoeol(win)

WINDOW  *win;
```

Return Values:

OK on success; ERR on failure.

cos Function

```
#include math
double cos(x)
double x;
```

Return Values:

Cosine of x.

cosh Function

```
#include math
double cosh(x)
double x;
```

Return Values:

Hyperbolic cosine of *x*.

creat Function

```
int  creat(file_spec, mode [,file_atts, . . . ])
char     *file_spec, *file_atts;
unsigned  mode;
```

Return Values:

Integer file descriptor, if the file is successfully created; –1, if an error occurs.

Notes:

See **chmod** for a description of the *mode* Argument.

file_atts Argument

"alq=n"	Decimal allocation quantity
"bls=n"	Decimal block size
"ctx=bin"	No translation of "\n' to terminal
"deq=n"	Decimal default extension quantity
"dna=file-spec"	Default filename string
"fop=val,val, . . . "	File processing options, where *val* is one of the following:

ctg	—	Contiguous
cbt	—	Contiguous-best-try
tef	—	Truncate at end-of-file

	cif	— Create if nonexistent
	sup	— Supersede
	scf	— Submit as command file on close
	spl	— Spool to system printer on close
	tmd	— Temporary delete
	tmp	— Temporary (no file directory)
	nef	— Not end-of-file
"fsz=n"	Decimal fixed header size	
"mbc=n"	Decimal multiblock count	
"mbf=n"	Decimal multibuffer count	

"mrs=n"	Decimal maximum record size
"rat=val,val, . . . "	Record attributes where *val* is one of the following:
	cr — Carriage-return control
	blk — Records to span block boundaries
	ftn — FORTRAN print control
	prn — Print file format
"rfm=val"	Record format, where *val* is one of the following:
	fix — Fixed-length record format
	stm — RMS stream record format
	stmlf — Stream format, line-feed terminator
	stmcr — Stream, carriage-return terminator
	var — Variable-length record format
	vfc — Variable-length with fixed control

	udf — Undefined
"rop=val"	Record processing, where *val* is one of the following:
	asy — Asynchronous I/O
	tmo — Timeout I/O

"shr=val"	File sharing, where *val* is one of the following: del — Allows users to delete get — Allows users to read mse — Allows mainstream access nil — Prohibits file sharing put — Allows users to write upd — Allows users to update upi — Allows one or more writers
"tmo=n"	Decimal I/O timeout value

[no]crmode Macros

```
#include curses

crmode()
nocrmode()
```

Return Values:

None.

Notes:

VAX C provides these macros only for portability; they have no functionality.

ctermid Function

```
#include stdio

char  *ctermid([string])
char  *string;
```

Return Values:

Name of the controlling terminal is returned to *string*. If no argument is given, the function returns the address of an internal storage area containing the string.

ctime Function

```
#include time

char  *ctime(bintim)

int   *bintim
```

Return Values:

Pointer to the time in the format: *wkd mmm dd hh:mm:ss* 19*yy*\n\0.

cuserid Function

```
#include stdio

char  *cuserid(string)

char  *string;
```

Return Values:

The name of the user who initiated the current process is returned to *string*. If no argument is given, the function returns the address of the name.

delch Macro and wdelch Function

```
#include curses

delch()
wdelch(win)

WINDOW  *win;
```

Return Values:

OK on success; ERR on failure.

delete Function

```
#include stdio

int   delete(file_spec)

char  *file_spec;
```

Return Values:

0, if the file is deleted; –1, if not.

deleteln Macro and wdeleteln Function

```
#include curses

deleteln()
wdeleteln(win)

WINDOW  *win;
```

Return Values:

OK on success; ERR on failure.

delwin Function

```
#include curses

delwin(win)

WINDOW  *win;
```

Return Values:

OK on success; ERR on failure.

dup and dup2 Functions

```
int  dup(file_desc_1)
int  dup2(file_desc_1, file_desc_2)

int  file_desc_1, file_desc_2;
```

Return Values:

A new file descriptor (**dup**) and a new descriptor pointing to the same file as *file_desc_1* (**dup2**); –1, if *file_desc_1* does not point to an open file or if the new file descriptor cannot be allocated.

dup2 Function

See **dup**.

[no]echo Macros

```
#include curses

echo()
noecho()
```

Return Values:

None.

ecvt, fcvt, and gcvt Functions

```
char  *ecvt(value, ndigit, decpt, sign)
char  *fcvt(value, ndigit, decpt, sign)
char  *gcvt(value, ndigit, buffer)

double value;
int    ndigit, *decpt, *sign;
char   *buffer;
```

Return Values:

ecvt and **fcvt**

The position of the decimal point relative to the first character in the string is returned by the *decpt* argument. A nonzero integer is returned to the *sign* argument if the input value is negative; otherwise, 0 is returned.

gcvt

The converted string is placed in the *buf* argument and the address of *buf* is returned.

endwin Function

```
#include curses

endwin()
```

Return Values:

OK on success; ERR on failure.

erase Macro and werase Function

```
#include curses

erase()
werase(win)

WINDOW  *win;
```

Return Values:

OK on success; ERR on failure.

execl, execle, execv, and execve Functions

```
int execl(name, arg0, . . . , argn, 0)
char *name, *arg0, . . . , *argn;

int execv(name, argv)
char *name, *argv[];
```

```
int execle(name, arg0, . . . , argn, 0, envp)
char *name, *arg0, . . . , *argn, *envp[];

int execve(name, argv, envp)
char *name, *argv[], *envp[];
```

Return Values:

–1 on failure.

Notes:

See **getenv** for a list of the strings in the *envp* environment array.

execle Function

See **execl**.

execv Function

See **execl**.

execve Function

See **execl**.

exit, and _exit Functions

```
exit([status])
_exit([status])

int status;
```

Return Values:

The process status is returned to the parent process, if any, or to the command language interpreter.

exp Function

```
#include math

double exp(x)

double x;
```

Return Values:

Base e raised to the power of the argument. If an overflow occurs, the function returns the largest possible floating-point value and sets the variable, errno, to ERANGE.

fabs Function

See **abs**.

fclose Function

```
#include stdio
int  fclose(file_ptr)
FILE *file_ptr;
```

Return Values:

0, if the file is successfully closed; –1, if not.

fcvt Function

See **ecvt**.

fdopen Function

```
#include stdio
#include file

FILE *fdopen(file_desc, a_mode)

int  file_desc;
char *a_mode;
```

feof Macro

```
#include stdio
int   feof(file_ptr)
FILE  *file_ptr;
```

Return Values:

Nonzero integer on end-of-file; 0, otherwise.

ferror Macro

```
#include stdio
int   ferror(file_ptr)
FILE  *file_ptr;
```

Return Values:

A nonzero integer if an error occurs. Subsequent calls to **ferror** continue to return this value until the file is closed or until **clearerr** is called.

fflush Function

```
#include stdio

int   fflush(file_ptr)

FILE  *file_ptr;
```

Return Values:

0, if the file is successfully flushed; EOF, if not.

fgetc and getw Functions, and getc Macro

```
#include stdio

int fgetc(file_ptr)
int getw(file_ptr)
int getc(file_ptr)

FILE *file_ptr;
```

Return Values:

The next character in the file (**fgetc** and **getc**) and the next four characters from the file (**getw**); EOF on error.

Notes:

Since EOF is defined in the *stdio* module to be an integer, use **feof** and **ferror** to check the success of **getw**.

fgetname Function

```
#include stdio

char  *fgetname(file_ptr, buffer [,style])

FILE  *file_ptr,
char  *buffer;
int   style;
```

Return Values:

Address of the buffer containing the file specification on success; 0 on error.

Notes:

The third argument, style, can be either one of the integers 1 or 0. If you specify 0, the function **fgetname** returns the file specification in VAX/VMS format. If you specify 1, the function **fgetname** returns the file specification in DEC/Shell format. If you do not specify this argument, this function returns the file name according to your current command language interpretor.

fgets Function

```
#include stdio

char *fgets(str, maxchar, file_ptr)

char  *str;
int   maxchar;
FILE  *file_ptr;
```

Return Values:

NULL on end-of-file; otherwise, the address of the first character in *string*.

fileno Macro

```
#include stdio
int  fileno(file_ptr)
FILE *file_ptr;
```

Return Values:

Integer file descriptor that identifies the file.

floor Function

```
#include math
double floor(x)
double x;
```

Return Values:

Largest integer that is less than or equal to *x*.

fopen Function

```
#include stdio
FILE  *fopen(file_spec, a_mode [,file_atts, . . .  ])
char  *file_spec,  *a_mode;
```

Return Values:

A file pointer for the named file, if the file was successfully opened; NULL on error.

Notes:

a_mode	Value
"r"	Read access
"w"	Write access
"a"	Append access
"r+"	Read update access
"w+"	Write update access
"a+"	Append update access

Notes:

For a description of *file_atts*, see **creat**.

fprintf and sprintf Functions

```
#include stdio

int fprintf(file_ptr, format_spec [,output_src, . . .  ])
int sprintf(str, format_spec[,output_src, . . .  ])

FILE  *file_ptr;
char  *str, *format_spec;
```

Return Values:

The number of characters written to the file (**fprintf**) or to the string (**sprintf**); –1 on error.

Notes:

format_spec	**Meaning**
d	Expect a decimal integer in the input. The corresponding argument must point to an **int**.
o	Expect an octal integer in the input (with or without a leading zero). The corresponding argument must point to an **int**.
x	Expect a hexadecimal integer in the input (without a leading 0x). The corresponding argument must point to an **int**.

format_spec	**Meaning**
c	Expect a character in the input. The corresponding argument must point to a **char**. The usual skipping of white-space characters can be disabled in this case, so that *n* white-space characters can be read with *%nc*. If a field width is given with c, the given number of characters is read and the corresponding argument should point to an array of **char**.
s	Expect a string in the input. The corresponding argument must point to an array of characters that is large enough to contain the string plus the terminating NUL character (\0). The input field is terminated by a space, tab, or newline.
e, f	Expect a floating-point number in the input. The corresponding argument must point to a **float**. The input format for floating-point numbers is [+|−]nnn[.ddd][(E|e)[+|−]nn], where the n's and the d's are decimal digits (as many as indicated by the field width minus the signs and the letter E).

ld, lo, lx	Same as d, o, and x, except that a **long** integer of the specified radix is expected. (Retained for portability only, since **long** and **int** are the same in VAX C.)
le, lf	Same as e, and f, except that the corresponding argument is a **double** instead of a **float**. The same effect can be achieved by using an uppercase E or F.
hd, ho, hx	Same as d, o, and x, except that a **short** integer of the specified radix is expected.
[...]	Expect a string that is not delimited by white-space characters. The brackets enclose a set of characters (not a string). Ordinarily, this set (or "character class") is made up of the characters that comprise the string field. Any character not in the set will terminate the field. However, if the first (leftmost) character is an up-arrow, then the set shows the characters that terminate the field. The corresponding argument must point to an array of characters.

fputc and putw Functions, and putc Macro

```
#include stdio

int fputc(character, file_ptr)
int putw(integer, file_ptr)
int putc(character, file_ptr)

ch     character;
int    integer;
FILE   *file_ptr;
```

Return Values:

The argument, *character* in the file (**fputc** and **putc**) the four characters in *integer* (**getw**); EOF on error.

Notes:

Since EOF is defined in the *stdio* module to be an integer, use **feof** and **ferror** to check the success of **putw**.

fputs Function

```
#include stdio

int   fputs(str, file_ptr)

char  *str;
FILE  *file_ptr;
```

Return Values:

Last character written; EOF on error.

fread Function

```
#include stdio

int fread(pointer, size_of_item, number_items, file_ptr)

int   number_items, size_of_item;
FILE  *file_ptr;
```

Return Values:

The number of items read; 0 on end-of-file or error.

free Function

See **cfree**.

freopen Function

```
#include stdio

FILE *freopen(file_spec, a_mode, file_ptr [,file_atts, . . . ])
```

```
char *file_spec, *a_mode;
FILE *file_ptr;
```

Return Values:

File pointer that points to the newly opened file; NULL on error.

Notes:

See also, **fopen**.

frexp Function

```
#include math

double frexp(value, eptr)

double value;
int    *eptr;
```

Return Values:

The mantissa of *value* with a magnitude less than 1. The exponent is returned to **eptr*.

fscanf and sscanf Functions

```
#include stdio

fscanf(file_ptr, format_spec [,input_ptr, . . .  ])
sscanf(str, format_spec [,input_ptr . . .  ])
```

```
FILE  *file_ptr;
char  *str, *format_spec;
```

Return Values:

The number of successfully matched and assigned input items; EOF on error or end-of-file.

Notes:

format_spec	Meaning
d	Convert to decimal format.
o	Convert to octal format.

format_spec	Meaning
x	Convert to unsigned hexadecimal format (without leading 0x). An uppercase X causes the hexadecimal digits A-F to be printed in uppercase. A lowercase x causes those digits to be printed in lowercase.
u	Convert to unsigned decimal format (giving a number in the range zero to 4,294,967,295).
c	Output single character (NUL characters are ignored.)
s	Write characters until NUL is encountered or until number of characters indicated by the precision specification is exhausted. If the precision specification is zero or omitted, all characters up to a NUL are output.
e	Convert **float** or **double** to the format [−]m.nnnnnnE[+|−]xx, where the number of n's is specified by the precision (default = 6) If the precision is explicitly zero, the decimal point appears but no n's appear. An E is printed if the conversion character is an uppercase E. An e is printed if the conversion character is a lowercase e.

f	Convert **float** or **double** to the format [–]m..m.nnnnnn, where the number of n's is specified by the precision (default - 6). Note that the precision does not determine the number of significant digits printed. If the precision is explicitly zero, no decimal point appears and no n's appear.
g	Convert **float** or **double** to d, e, or f format, whichever is shorter (suppress insignificant zeros).
%	Write out the percent symbol. No conversion is performed.

The following characters can be used between the percent sign (%) and the conversion character. They are optional, but if specified, they must occur in the order listed.

Character	Meaning
-	Left-justify the converted output source in its field.
width	Use this integer constant as the minimum field width. If the converted output source is wider than this minimum, write it out anyway. If the converted output source is narrower than the minimum width, pad it to make up the field width. Padding is with spaces normally, and with zeros if the field width is specified with a leading zero; this does not mean that the width is an octal number. Padding is on the left normally and on the right if a minus sign is used.
.	Separates field width from precision.
precision	Use this integer constant to designate the maximum number of characters to print with s format, or the number of fractional digits with e or f format.

l	Indicates that a following d, o, x, or u specification corresponds to a **long** output source. In VAX C, all **int** values are long by default.
*	Can be used to replace the field width specification and/or the precision specification. The corresponding width or precision is given in the output source.

fseek Function

```
#include stdio

int fseek(file_ptr, offset, direction)

FILE  *file_ptr;
int   offset, direction;
```

Return Values:

0 for successful seeks; EOF on error.

Notes:

direction	Value
0	From the beginning of the file
1	From the current position
2	From the end-of-file

fstat and stat Functions

```
#include stat

fstat(file_desc, buffer)
stat(file_spec, buffer)

int          file_desc;
char         *file_spec;
struct stat  *buffer;
```

Return Values:

0 on success; –1 on error.

ftell Function

```
#include stdio
int  ftell(file_ptr)
FILE  *file_ptr;
```

Return Values:

Byte offset from the beginning of the file to the current location within the file; –1 on error.

ftime Function

```
#include timeb
ftime   (time_pointer)
struct  timeb  *time_pointer;
```

Return Values:

The number of seconds that have elapsed on the system since 00:00:00 January 1, 1970 is returned to the structure, timeb.

fwrite Function

```
#include stdio

int fwrite(ptr, size_of_item, number_items, file_ptr)

int   number_items, size_of_item;
FILE  *file_ptr;
```

Return Values:

The number of items written; 0 on error.

gcvt Function

See **cvt**.

getc Macro

See **fgetc**.

getch Macro and wgetch Function

```
#include curses

getch()
wgetch(win)

WINDOW  *win;
```

Return Values:

The character from the window; ERR, if the screen scrolls illegally.

getchar Function

```
int  getchar()
```

Return Values:

The next character from the standard input device (stdin, the terminal); EOF on error.

getegid, geteuid, getgid, and getuid Functions

```
unsigned  getegid()
unsigned  geteuid()
unsigned  getgid()
unsigned  getuid()
```

Return Values:

The group number from the UIC (**getgid** and **getegid**) or the member number from the UIC (**getuid** and **geteuid**).

Notes:

The functions **getuid** and **geteuid** are functionally equivalent in VAX C.

getenv Function

```
char  *getenv(name)
char  *name;
```

Return Values:

The function returns one of the following values depending on the value of the argument, name, specified in the function call:

Argument	Return Value
HOME	The user's login directory
TERM	The terminal type
PATH	The default device and directory
USER	The name of the user initiating the process

geteuid Function

See **getegid**.

getgid Function

See **getegid**.

getname Function

```
char *getname(file_desc, buffer [,style])

int  file_desc, style;
char *buffer;
```

Return Values:

The address of the buffer containing the file specification; -1 on error.

Notes:

The third argument, style, can be either one of the integers 1 or 0. If you specify 0, the function **fgetname** returns the file specification in VAX/VMS format. If you specify 1, the function **fgetname** returns the file specification in DEC/Shell format. If you do not specify this argument, this function returns the file name according to your current command language interpretor.

getpid Function

```
int  getpid()
```

Return Values:

The current process ID.

gets Function

```
#include stdio

char *gets(str)

char *str;
```

Return Values:

A pointer to the character string containing the line; NULL, if end-of-file is reached before the newline is encountered or on error.

getstr Macro and wgetstr Function

```
#include curses

getstr(str)
wgetstr(win, str)
```

```
WINDOW  *win;
char    *str;
```

Return Values:

OK on success; ERR, if the screen scrolls illegally.

getuid Function

See **getegid**.

getw Function

See **fgetc**.

getyx Macro

```
#include curses

getyx(win, y, x)

WINDOW  *win;
int     y, x;
```

Return Values:

OK on success; ERR on failure.

gsignal Function

```
#include signal

int  gsignal(sig [,code]))

int  sig;
```

Return Values:

If **gsignal** specifies a sig argument that is outside the range defined in the signal module, then **gsignal** returns zero, and the variable, errno, is set to EINVAL.

If **ssignal** establishes SIG_DFL (default action) for the signal, then **gsignal** does not return. The image is exited with the VAX/VMS error code that corresponds to the signal.

If **ssignal** establishes SIG_IGN (ignore signal) as the action for the signal, then **gsignal** returns its sig.

Otherwise, **ssignal** must have established an action function for the signal. That function is called, and that function's return value is returned by **gsignal**.

Notes:

Hardware Condition	Signal	Code
Integer overflow	SIGFPE	FPE_INTOVF_TRAP
Integer division by zero	SIGFPE	FPE_INTDIV_TRAP
Floating overflow trap	SIGFPE	FPE_FLTOVF_TRAP
Floating/decimal division by zero	SIGFPE	FPE_FLTDIV_TRAP
Floating underflow trap	SIGFPE	FPE_FLTUND_TRAP
Decimal overflow trap	SIGFPE	FPE_DECOVF_TRAP
Subscript-range	SIGFPE	FPE_SUBRNG_TRAP
Floating overflow fault	SIGFPE	FPE_FLTOVF_FAULT
Floating divide by zero fault	SIGFPE	FPE_FLTDIV_FAULT
Floating underflow fault	SIGFPE	FPE_FLTUND_FAULT

Reserved instruction	SIGILL	ILL_PRIVIN_FAULT
Reserved operand	SIGILL	ILL_RESOP_FAULT
Reserved addressing	SIGILL	ILL_RESAD_FAULT
Compatibility mode	SIGILL	Hardware supplied
Length access control	SIGSEGV	—
Chme	SIGSEGV	—
Chms	SIGSEGV	—
Chmu	SIGSEGV	—
Trace pending	SIGTRAP	—

Hardware Condition	Signal	Code
Bpt instruction	SIGTRAP	—
Protection violation	SIGBUS	—
Customer-reserved code	SIGEMT	—

hypot Function

See **cabs**.

inch Macro and winch Function

```
#include curses

inch()
winch(win)

WINDOW  *win;
```

Return Values:

OK on success; ERR on failure.

initscr Function

```
#include curses

initscr()
```

Return Values:

OK on success; ERR on failure.

insch Macro and winsch Function

```
#include curses

insch(ch)
winsch(win, ch)

WINDOW  *win;
char     ch;
```

Return Values:

OK on success; ERR, if the screen scrolls illegally.

insertln Macro and winsertln Function

```
#include curses

insertln()
winsertln(win)

WINDOW  *win;
```

Return Values:

OK on success; ERR, if the screen scrolls illegally.

insstr Macro and winsstr Function

```
#include curses

insstr(str)
winsstr(win, str)

WINDOW  *win;
char    *str;
```

Return Values:

OK on success; ERR, if the screen scrolls illegally.

Notes:

This function and macro are VAX C specific.

isalnum Macro

```
#include ctype

int  isalnum(character)

char character;
```

Return Values:

A nonzero integer, if the character is alphanumeric; 0, if it is not.

isalpha Macro

```
#include ctype

int  isalpha(character)

char character;
```

Return Values:

A nonzero integer, if the character is alphabetic; 0, if it is not.

isapipe Function

```
int  isapipe(file_desc)

int  file_desc;
```

Return Values:

1, if the specified file descriptor is associated with a mailbox; 0, if it is not; –1 on error.

isascii Macro

```
#include ctype
int  isascii(character)
char character;
```

Return Values:

A nonzero integer, if the character is ASCII; 0, if it is not.

isatty Function

```
int  isatty(file_desc)
int  file_desc;
```

Return Values:

1, if the file is a terminal; 0, if the file is not.

iscntrl Macro

```
#include ctype
int  iscntrl(character)
char character;
```

Return Values:

A nonzero integer, if the character is a control character; 0, if not.

isdigit Macro

```
#include ctype
int  isdigit(character)
char character;
```

Return Values:

A nonzero integer, if the character is a digit; 0, if it is not.

isgraph Macro

```
#include ctype
int  isgraph(character)
char character;
```

Return Values:

A nonzero integer, if the character is an ASCII graphic character; 0, if it is not.

Notes:

This macro is VAX C specific.

islower Macro

```
#include ctype
int  islower(character)
```

```
char character;
```

Return Values:

A nonzero integer, if the character is lowercase; 0, if it is not.

isprint Macro

```
#include ctype
int  isprint(character)
char character;
```

Return Values:

A nonzero integer, if the character is an ASCII printing character; 0, if it is not.

ispunct Macro

```
#include ctype
int  ispunct(character)
char character;
```

Return Values:

A nonzero integer, if the character is a punctuation character; 0, if it is not.

isspace Macro

```
#include ctype
int  isspace(character)
char character;
```

Return Values:

A nonzero integer, if the character is one of the whitespace characters; 0, if it is not.

isupper Macro

```
#include ctype

int  isupper(character)

char character;
```

Return Values:

A nonzero integer, if the character is uppercase; 0, if it is not.

isxdigit Macro

```
#include ctype
int  isxdigit(character)
char character;
```

Return Values:

A nonzero integer, if the character is a hexadecimal digit; 0, if it is not.

Notes:

This macro is VAX C specific.

kill Function

```
int  kill(pid, sig)
int  pid, sig;
```

Return Values:

0, if the signal is successfully queued; –1, if an error occurs.

Notes:

The receiving process always terminates.

ldexp Function

```
#include math

double ldexp(x, e)

double x;
int    e;
```

Return Values:

The first argument times 2 to the power of its second argument ($x(2^{e})$) . If underflow occurs, this function returns 0. If overflow occurs, it returns the largest possible value of the appropriate sign.

leaveok Macro

```
#include curses

leaveok(win, boolf)
```

```
WINDOW  *win;
bool    boolf;
```

Return Values:

OK on success; ERR on failure.

localtime Function

```
#include time

struct  tm  *localtime(bintim)

int  *bintim;
```

Return Values:

A pointer to the structure with the tag, tm. The members of the tm structure are integers representing the following:

Member		Description
tm_sec	—	time in seconds
tm_min	—	minutes
tm_hour	—	hours (24)

tm_mday	—	day of the month (1-31)
tm_mon	—	month (0-11)
tm_year	—	year (last two digits)
tm_wday	—	day of the week (0-6)
tm_yday	—	day of the year (0-365)
tm_isdst	—	daylight savings time (always 0)

log and log10 Function

```
#include math

double log(x)
double log10(x)

double x;
```

Return Values:

The natural base-e (**log**) and the base-10 (**log10**) logarithm of its argument. If the argument, x, is 0 or negative, the function returns 0 and sets the variable, errno, to EDOM.

log10 Function

See **log**.

longjmp and setjmp Functions

```
#include setjmp

setjmp(env)
longjmp(env, val)

jmp_buf  env;
int      val;
```

Return Values:

The function **longjmp** returns *val* to the **setjmp** function with which it is associated.

When the function **setjmp** is first called, it returns 0. After **longjmp** is called with the same *env* argument as the first **setjmp** call, **setjmp** returns the value of the **longjmp** call's *val* argument.

longname Function

```
longname(termbuf, name)
char     *termbuf, *name;
```

Return Values:

The full terminal name is placed in the argument, name.

Notes:

This function has limited functionality in VAX C.

lseek Function

```
int  lseek(file_desc, offset, direction)
int  file_desc, offset, direction;
```

Return Values:

The new position in the file. If the file descriptor is undefined or if you try to seek before the beginning of the file, the function returns –1.

Notes:

direction	Value
0	From the beginning of the file
1	From the current position
2	From the end-of-file

mkdir Function

```
int  mkdir(dir_spec, mode [,uic, max_versions, r_v_num])

char      *dir_spec;
unsigned  mode, uic, max_version, r_v_num;
```

Return Values:

0 on success; –1 on error.

malloc Function

See **calloc**.

mktemp Function

```
#include stdio

char *mktemp(template)

char *template;
```

Return Values:

A pointer to a temporary file name created from a *template* of the form "[*nam*]XXXXXX". If a unique file name cannot be created, **mktemp** returns a pointer to an empty string.

modf Function

```
#include math

double modf(value, iptr)

double value, *iptr;
```

Return Values:

The positive fractional part of *value* and the address of the integral part is assigned to *iptr*.

move Macro and wmove Function

```
#include curses

move(y, x)
wmove(win, y, x)
```

```
WINDOW  *win;
int     y, x;
```

Return Values:

OK on success; ERR, if the screen scrolls illegally.

mv[w]addch Macros

```
#include curses

mvaddch(y, x, ch)
mvwaddch(win, y, x, ch)

WINDOW  *win;
char      ch;
int     y, x;
```

Return Values:

OK on success; ERR, if the screen scrolls illegally.

mv[w]addstr Macros

```
#include curses

mvaddstr(y, x, str)
mvwaddstr(win, y, x, str)

WINDOW  *win;
char    *str;
int     y, x;
```

Return Values:

OK on success; ERR, if the screen scrolls illegally.

mvcur Function

```
#include curses

mvcur(lasty, lastx, newy, newx)

int  lasty, lastx, newy, newx;
```

Return Values:

OK on success; ERR on failure.

mv[w]delch Macros

```
#include curses

mvdelch(y, x)
mvwdelch(win, y, x)

WINDOW  *win;
int     y, x;
```

Return Values:

OK on success; ERR, if the screen scrolls illegally.

mv[w]getch Macros

```
#include curses

mvgetch(y, x)
mvwgetch(win, y, x)

WINDOW  *win;
int     y, x;
```

Return Values:

OK on success; ERR on failure.

mv[w]getstr Macros

```
#include curses

mvgetstr(y, x, str)
mvwgetstr(win, y, x, str)

WINDOW  *win;
char    *str;
int     y, x;
```

Return Values:

OK on success; ERR on failure.

mv[w]inch Macros

```
#include curses

mvinch(y, x)
mvwinch(win, y, x)

WINDOW  *win;
int     y, x;
```

Return Values:

OK on success; ERR on failure.

mv[w]insch Macros

```
#include curses

mvinsch(ch, y, x)
mvwinsch(win, y, x, ch)

WINDOW  *win;
char     ch;
int      y, x;
```

Return Values:

OK on success; ERR, if the screen scrolls illegally.

mv[w]insstr Macros

```
#include curses

mvinsstr(y, x, str)
mvwinsstr(win, y, x, str)

WINDOW  *win;
char    *str;
int     y, x;
```

Return Values:

OK on success; ERR, if the screen scrolls illegally.

mvwin Function

```
#include curses

mvwin(win, y, x)

WINDOW  *win;
int     y, x;
```

Return Values:

OK on success; ERR, if moving the window puts all or part of the window off of the terminal screen. On error, this function does not attempt to move the window and the screen remains unaltered.

newwin Function

```
#include curses

newwin(numlines, numcols, begin_y, begin_x)

int numlines, numcols, begin_y, begin_x;
```

Return Values:

A pointer to a newly created window; ERR on failure.

nice Function

```
int  nice(increment)
int  increment;
```

Return Values:

0, if the process priority is successfully lowered; −1, if it is not.

[no]nl Macros

```
#include curses

nl()
nonl()
```

Return Values:

None.

Notes:

These macros have no functionality in VAX C.

open Function

```
#include file

int  open(file_spec, flags, mode [,file_attribute, . . .  ])

char *file_spec;
int  flags, mode;
```

Return Values:

An integer file descriptor, if the file is successfully opened; –1, if it is not.

Notes:

The values for the second argument, flags:

O_RDONLY	Open for reading only
O_WRONLY	Open for writing only
O_RDWR	Open for reading and writing
O_NDELAY	Ignored; not supported by VAX C
O_APPEND	Append on each write
O_CREAT	Create a file if it does not exist
O_TRUNC	Create a new version of this file
O_EXECL	Error if attempting to create existing file

overlay Function

```
#include curses
overlay(win1, win2)
WINDOW  *win1, *win2;
```

Return Values:

OK on success; ERR on failure.

overwrite Function

```
#include curses
overwrite(win1, win2)
WINDOW  *win1, *win2;
```

Return Values:

OK on success; ERR on failure.

pause Function

```
pause()
```

Return Values:

None.

perror Function

```
#include perror

perror(string)

char  *string;
```

Return Values:

A message to stdout (the terminal) of the form: *string: message*\n

pipe Function

```
#include file

int pipe(array_fdscpt [, flags, bufsize])

int array_fdscptr[2];
int flags, bufsize;
```

Return Values:

0, if the pipe is successfully created; –1, if not.

Notes:

If you do not specify an argument for *bufsize*, the default size of the created mailbox buffer is 512 bytes. For information on the argument, flags, see **open**. You only need to include the *file* module if you specify *flags*.

pow Function

```
#include math

double pow(x, y)

double x, y;
```

Return Values:

The argument, *x*, to the power of *y*.

If the result overflows, the function returns the largest possible floating-point value and sets the variable, errno, to ERANGE.

If *y* is negative or nonintegral, or if both arguments are 0, the function returns 0.

printf Function

```
#include stdio
int  printf(format_spec [,output_src . . . ])
char *format_spec;
```

Return Values:

The number of characters written; –1 on error.

Notes:

For information on *format_spec*, see **fprintf**.

[w]printw Functions

```
#include curses
```

```
printw(format_spec [,output_src, . . . ])
wprintw(win, format_spec [,output_src, . . . ])
WINDOW *win;
```

Return Values:

OK on success; ERR, if the screen scrolls illegally.

Notes:

For information on *format_spec*, see **fprintf**.

putc Macro

See **fputc**.

putchar Function

```
int  putchar(character)
char character;
```

Return Values:

The character written; EOF on failure.

puts Function

```
#include stdio
int  puts(str)
char *str;
```

Return Values:

0 if the *string* was written to stdout (the terminal); EOF on failure.

putw Function

See **fputc**.

rand and srand Functions

```
int rand()
int srand(seed)
int seed;
```

Return Values:

Pseudorandom numbers in the range *0* to $2^{31}-1$.

[no]raw Macros

```
#include curses

raw()
noraw()
```

Return Values:

None.

read Function

```
int  read(file_desc, buffer, nbytes)

int  file_desc, nbytes;
char *buffer;
```

Return Values:

The number of bytes read; 0, if end-of-file is reached; −1 on error.

realloc Function

```
char *realloc(pointer, size)

char      *pointer;
unsigned  size;
```

Return Values:

The address of the area; 0 on error.

refresh Macro and wrefresh Function

```
#include curses

refresh()
wrefresh(win)

WINDOW  *win;
```

Return Values:

OK on success; ERR on failure.

rewind Function

```
#include stdio

int   rewind(file_ptr)

FILE  *file_ptr;
```

Return Values:

0, if the file is successfully rewound; –1, if an error occurs.

sbrk Function

See **brk**.

scanf Function

```
#include stdio
int  scanf(format_spec [,input_ptr . . .  ])
char *format_spec;
```

Return Values:

The number of successfully scanned items; EOF on end-of-file.

Notes:

For information concerning *input_ptr*, see **fscanf**.

scanw Macro and wscanw Function

```
#include curses

scanw(fmt_spec [,input_ptr, . . . ])
wscanww(win, fmt_spec [,input_ptr, . . . ])

WINDOW  *win;
```

Return Values:

OK on success; ERR, if the screen scrolls illegally.

scroll Function

```
#include curses

scroll(win)

WINDOW  *win;
```

Return Values:

OK on success; ERR on failure.

scrollok Macro

```
#include curses

scrollok(win, boolf)

WINDOW  *win;
bool    boolf;
```

Return Values:

OK on success; ERR on failure.

setattr Macro and wsetattr Function

```
#include curses

setattr(attr)
wsetattr(win, attr)

WINDOW  *win;
```

Return Values:

OK on success; ERR on failure.

Notes:

attr Argument

_BLINK	Clear blinking attribute
_BOLD	Clear bold face attribute
_REVERSE	Clear reverse video attribute
_UNDERLINE	Clear underline attribute

This function and macro are VAX C specific.

setbuf Function

```
#include stdio

setbuf(file_ptr, buffer)

FILE *file_ptr;
char *buffer;
```

Return Values:

None.

setgid and setuid Functions

```
int  setgid(group_number)
int  setuid(member_number)

unsigned  member_number;
unsigned  group_number;
```

Return Values:

None.

setjmp Function

See **longjmp**.

setuid Function

See **setgid**.

sigblock Function

```
sigblock(mask)
int   mask;
```

Return Values:

The previous set of masked signals; –1 on error.

signal Function

```
#include signal

int (*signal(sig, func))()

int sig;
int (*func)();
```

Return Values:

The address of the function previously (or initially) established to handle the signal. If the argument, sig, is out of range, this function returns –1 and the variable, errno, is set to EINVAL.

Notes:

Signal symbolic names:

Name	Description	Generated by
SIGHUP	Hang up	Data set hang up
SIGINT	Interrupt	VAX/VMS CTRL/C interrupt
SIGQUIT	Quit	CTRL/C if the action for SIGINT is SIG_DFL (default)
SIGILL[1]	Bad instruction	Illegal instruction, reserved operand, or reserved address mode
SIGTRAP[1]	Trace trap	TBIT trace trap or breakpoint fault instruction
SIGIOT	IOT instruction	Not implemented
SIGEMT	EMT instruction	Compatibility mode trap or op code reserved to customer
SIGFPE	Floating-point	Floating-point overflow

SIGKILL[2]	Kill	External signal only
SIGBUS	Bus error	Access violation or change mode user
SIGSEGV	Segment violation	Length violation or change mode supervisor
SIGSYS	Call error	Bad argument to system call
SIGPIPE	Broken pipe	Not implemented
SIGALRM	Alarm clock	Timer AST
SIGTERM	Software	External signal only

[1] Not reset when caught.

[2] Cannot be caught or ignored.

sigpause Function

```
sigpause(mask)
int   mask;
```

Return Values:

After restoring the previous set of masked signals, this function returns EINTR which causes an interrupt; –1 on error.

sigsetmask Function

```
sigsetmask(mask)
int  mask;
```

Return Values:

The previous set of masked signals; –1 on error.

sigstack Function

```
#include signal

sigstack(ss, oss)

struct sigstack
   {
      char     *ss_sp;
      int      ss_onstack;
   };
struct  sigstack  *ss,  *oss;
```

Return Values:

0 on success; –1 on failure.

sigvec Function

```
#include signal

sigvec(sigint, sv, osv)

struct sigvec
   {
      int   (*handler)();
      int   mask;
      int   onstack;
   };
struct  sigvec *sv, *osv;
int                sigint;

handler(sigint, code, scp)

int       sigint, code;
struct sigcontext *scp;
```

Return Values:

0 if the call to the signal handler is successful; –1 on error.

sin Function

```
#include math
double sin(x)
double x;
```

Return Values:

The sine of *x*.

sinh Function

```
#include math

double sinh(x)

double x;
```

Return Values:

The hyperbolic sine of its argument. Both *x* and its sine must be of type **double**. On overflow error, this function returns a **double** value with the largest possible magnitude and appropriate sign.

sleep Function

```
int  sleep(seconds)

unsigned  seconds;
```

Return Values:

The number of seconds that the process slept; –1 on error.

sprintf Function

See **fprintf**.

sqrt Function

```
#include math

double sqrt(x)

double x;
```

Return Values:

The square root of x; 0, if x is negative.

srand Function

See **rand**.

sscanf Function

See **fscanf**.

ssignal Function

```
#include signal

int (*ssignal(sig, func))()

int  sig,  (*func)();
```

Return Values:

The address of the function previously (or initially) established as the action for the signal; 0, if the previous action was SIG_DFL.

Notes:

See also, **signal**.

standend Macro and wstandend Function

```
#include curses

standend()
wstandend(win)

WINDOW  *win;
```

Return Values:

OK on success; ERR on failure.

standout Function

```
#include curses

standout()
wstandout(win)

WINDOW  *win;
```

Return Values:

OK on success; ERR on failure.

stat Function

See **fstat**.

strcat and strncat Function

```
char *strcat(str_1, str_2)
char *strncat(str_1, str_2, maxchar)
```

```
char *str_1, *str_2;
int  maxchar;
```

Return Values:

The address of *str_1*.

Notes:

The argument, maxchar, specifies the maximum number of characters to concatenate from *str_2* unless the NUL terminator is encountered first.

strchr and strrchr Functions

```
char  *strchr(str, character)
char  *strrchr(str, character)

char  *str, character;
```

Return Values:

The address of the first occurrence (**strchr**) or the last occurrence (**strrchr**) of *character* in the string; 0, if the character was not found.

strcmp and strncmp Functions

```
int  strcmp(str_1, str_2)
int  strncmp(str_1, str_2, maxchar)

char *str_1, *str_2;
int  max;
```

Return Values:

A negative, 0, or positive integer indicating whether *str_1* is composed of more, equal, or less characters than *str_2*.

Notes:

The argument, maxchar, specifies a maximum number of characters to search in both *str_1* and *str_2*.

(−) str_1 [redacted] str_2 <

(0) str_1 = str_2

(+) string_1 [redacted] str_2 >

strcpy and strncpy Functions

```
char  *strcpy(str_1, str_2)
char  *strncpy(str_1, str_2, maxchar)

char *str_1, *str_2;
int  max;
```

Return Values:

The address of *str_1*.

strcpt(dest, source)
strncpy (dest, sourcep)

Notes:

The argument, maxchar, specifies the maximum number of characters to copy from *str_2* to *str_1* up to but not including the NUL terminator.

strcspn Function

```
int  strcspn(str, charset)
char *str, *charset;
```

Return Values:

The number of characters preceding the first character in *str* that is also in *charset*. If no match is found, the function returns the length of *str*. This function returns 0 if *str* is null.

strlen Function

```
int  strlen(str)
char *str;
```

Return Values:

The length of *str*.

strncat Function

See **strcat**.

strncmp Function

See **strcmp**.

strncpy Function

See **strcpy**.

strpbrk Function

```
char  *strpbrk(str, charset)
char  *str, *charset;
```

Return Values:

The address of the first character in *str* that is also in *charset*; NULL, if no match is found.

strrchr Function

See **strchr**.

strspn Function

```
int  strspn(str, charset)
char *str, *charset;
```

Return Values:

The number of characters that precede the first character in *str* that is not also in *charset*. If *charset* is a null string, the function returns 0. If all the characters in *str* are also in *charset*, the function returns the length of *str*.

subwin Function

```
#include curses
subwin(win, numlines, numcols, begin_y, begin_x)
WINDOW  *win;
int numlines, numcols, begin_y, begin_x;
```

Return Values:

A pointer to a newly created subwindow; ERR on failure.

tan Function

```
#include math

double tan(x)

double x;
```

Return Values:

The tangent of *x*. At its singular points (. . . , *-3pi/2* , *-pi/2* , *pi/2* , . . .), the return value is the largest possible **double** value, and the variable, errno, is set to ERANGE

tanh Function

```
#include math

double tanh(x)

double x;
```

Return Values:

The hyperbolic tangent of *x*.

time Function

```
long  time(time_location)

long  *time_location;
```

Return Values:

The elapsed system time since 00:00:00 January 1, 1970. If the argument, time_location, is specified, it points to the location of the returned time.

times Function

```
times(buffer)

struct  tbuffer  *buffer;
```

Return Values:

The accumulated times of the current process and of its terminated child processed. The times are placed in the user-defined structure with the tag, tbuffer. The structure should have the following members of type **int**: *proc_user_time*, *proc_system_time*, *child_user_time*, and *child_system_time*. All system times are returned as 0. Accumulated CPU times are returned in 10-millisecond units.

tmpfile Function

```
#include stdio

FILE  *tmpfile()
```

Return Values:

A file pointer to the temporary file; a null pointer on error.

tmpnam Function

```
#include stdio

char *tmpnam(name)
char *name;
```

Return Values:

If *name* is specified, the function returns the file name string to *name*, or, if no argument is given, it returns the address of an internal storage area containing the string.

toascii Function

```
#include ctype
int  toascii(character)
char character;
```

Return Values:

The *character* converted to 7-bit ASCII.

tolower Function and _tolower Macro

```
#include ctype

char  tolower(character)
char  _tolower(character)

char  character;
```

Return Values:

The *character* converted to lowercase. Lowercase input characters are returned unchanged.

Notes:

_tolower is implemented as a macro.

touchwin Function

```
#include curses

touchwin(win)

WINDOW  *win;
```

Return Values:

OK on success; ERR on failure.

ttyname Function

```
#include curses

int   *ttyname()
```

Return Values:

A pointer to the NUL-terminated pathname of the terminal device associated with the file descriptor 0, stdin (the terminal).

Notes:

This function has limited functionality in VAX C.

toupper Function and _toupper Macro

```
#include ctype

char toupper(character)
char _toupper(character)

char character;
```

Return Values:

The *character* converted to uppercase. Uppercase input characters are returned unchanged.

Notes:

_toupper is implemented as a macro.

umask Function

```
int  umask(mode_complement)
unsigned  mode_complement;
```

Return Values:

The **chmod** argument, mode, corresponds to the argument, mask, except that mask has the effect of denying the specified privileges.

ungetc Function

```
#include stdio
int ungetc(character, file_ptr)
char  character;
FILE  *file_ptr;
```

Return Values:

The next character to be read (by **getc**); EOF on error.

va_arg Macro

```
#include varargs

va_arg(list_incrementor, item_type)

va_list  list_incrementor;
```

Return Values:

The next argument in the argument list.

va_count Macro

```
#include varargs
va_count(count)
int   count;
```

Return Values:

The number of longwords in the argument list, in the argument, count.

Notes:

This macro is VAX C specific.

va_end Macro

```
#include varargs
va_end(list_incrementor)
va_list  list_incrementor;
```

Return Values:

None.

va_start and va_start_1 Macros

```
#include varargs

int*    va_start(list_incrementor)
int*    va_start_1(list_incrementor, offset)

va_list  list_incrementor;
int      offset;

function_name(va_alist)
va_dcl
{
   va_list  list_incrementor;
      .
      .
      .
```

Return Values:

These macros initialize the argument, list_incrementor, to the first argument in the list.

Notes:

Use **va_start_1** when you need to access a variable-length argument list, that is itself a single argument in a larger list, and is preceded by a known number of defined arguments. For example, a VAX C RTL function which contains a variable-length argument list offset from the beginning of the entire argument list is **printf**. The macro **va_start_1** is VAX C specific.

va_start_1 Function

See **va_start**.

VAXC$ESTABLISH Function

```
void  VAXC$ESTABLISH (exception_handler)
int   (*exception_handler)();
```

Return Values:

Establishes the *exception_handler* as a legitimate one. Only condition handlers declared in this way should be used in VAX C programs. In this way **VAXC$ESTABLISH** catches all RTL-related exceptions and passes on all others to the declared handler.

vfork Function

```
int vfork()
```

Return Values:

0 to the child process and the child process ID to the parent process.

wait Function

```
int  wait([status])
int  *status;
```

Return Values:

The process ID of the terminated child process; –1, if there are no child processes.

wrapok Macro

```
#include curses

wrapok(win, boolf)

WINDOW *win;
bool boolf;
```

Return Values:

None.

Notes:

This macro has no functionality in VAX C.

write Function

```
int  write(file_desc, buffer, nbytes)

int  file_desc, nbytes;
char *buffer;
```

Return Values:

The number of bytes written; –1 on error.

_exit Function

See **exit**.

_tolower Macro

See **tolower**.

_toupper Macro

See **toupper**.

digital

AA-P788B-TE