main ()
{
    int c, i;
    long int nwhite, nother;

    nwhite = 0;
    nother = 0;
    for (i = 0; i < 10; i++)
        ndigit[i] = 0;

    while ((c = getchar()) != EOF)
    {
        if ('0' <= c && c <= '9')
            nwhite += 1;
        else if (c == ' ' || c == 'n' || c == '
')
            nother += 1;
    }

    printf("nwhite space = %d\n", nwhite);
    return(0);
}
C
LANGUAGE
PROGRAMMER'S
COMPANION

REVISION 19.3
FDR8419-193

This document reflects the software
as of Master Disk Revision 19.3.

by
Ron Johnson

Prime Computer, Inc.
500 Old Connecticut Path
Framingham, Massachusetts 01701
The Programmer's Companion is a series of pocket-size, quick-reference guides to Prime software products.

Published by Prime Computer, Inc.
Technical Publications Department
500 Old Connecticut Path
Framingham, MA 01701

Copyright © 1984 by Prime Computer, Inc.

Printed in USA. All rights reserved.

The Programmer's Companion and PRIMOS are registered trademarks of Prime Computer, Inc.

UNIX™ is a registered trademark of Bell Laboratories, Inc.

The information contained in this document is subject to change without notice and should not be construed as a commitment by Prime Computer. Prime Computer, Inc. assumes no responsibility for errors that may appear in this document.

Credits
Editorial Development       Betty B. Hoskins
Project Support             Anne Ladd
Design                      Susan Windheim
Production                  Leo Maldonado
Document Preparation        Gene Muir
Typesetter                  The Type Shop
Printer                     Winthrop Printing Company Inc.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Format Conventions</td>
<td>1</td>
</tr>
<tr>
<td>Glossary of Prime Concepts</td>
<td>3</td>
</tr>
<tr>
<td>C Language Concepts</td>
<td>7</td>
</tr>
<tr>
<td>Precedence and Associativity</td>
<td>14</td>
</tr>
<tr>
<td>Compiling and Loading C Programs</td>
<td>15</td>
</tr>
<tr>
<td>C Library Functions</td>
<td>21</td>
</tr>
<tr>
<td>Input Format Specifications</td>
<td>73</td>
</tr>
<tr>
<td>Output Format Specifications</td>
<td>76</td>
</tr>
<tr>
<td>Conversion Charts</td>
<td>79</td>
</tr>
<tr>
<td>ASCII Character Set</td>
<td>81</td>
</tr>
</tbody>
</table>

---

**Note**

This document responds to user requests for more information on Prime's implementation of the C programming language. The *C Programmer's Companion* contains the information needed to execute C language programs on a PRIMOS operating system. The *C User's Guide* (DOC7534-193) contains detailed descriptions of the library functions presented in this document.

**Printing History**

First Printing, June 1984
COMMAND FORMAT CONVENTIONS

Uppercase: Identifies command words, compiler options, or standard identifiers. Text must be entered literally.

ASSIGN
-64V
ABORT$ 

Lowercase: Identifies C library functions. Also identifies arguments for which you must substitute an appropriate numerical or text value.

mkdir()
-BINARY n

Rust Color: Indicates user input in programming examples.

OK, BIND

Abbreviations: Are indicated by rust-colored letters.

-NOCOMPATIBILITY
Square Brackets [ ]: Indicate an optional keyword or argument.

-LISTING [pathname]

[YES
[NO

Hyphen -: Identifies a PRIMOS command line option.

-CHECKOUT

Ellipsis . . .: Indicates that the preceding argument may be repeated.

COMMAND [argument-1] [argument-2] . . .

[argument-n]

Parentheses ( ): Indicate that an argument or a keyword must be entered literally.

fdtm(pathname)
64V Mode: See V Mode.

Access Category: A PRIMOS file system object that contains an access control list.

Access Control List (ACL): A list of users and their respective access rights to a PRIMOS file system object. See the Prime User's Guide (DOC4130-190) for more information on ACLs.

Binary File: A translation of a source file generated by a language translator (such as C, PL1/G, COBOL). Binary files are also commonly referred to as object files.

Byte: 8 bits or 1 character.

Condition Mechanism: A hardware feature of Prime machines. The condition mechanism directs a condition that is incapable of being handled by the software (such as division by zero or use of the BREAK signal) to a set of user-written routines. These user-written routines are known as on-units. The condition mechanism and the subroutines used to invoke it are described in the Subroutines Reference Guide (DOC3621-190).

Filename: The name of a file or a directory. PRIMOS filenames can contain up to 32 characters. The first character, however, must not be a digit (0–9).
**File System Error Code:** All Prime file system subroutines return numerical error codes. For example, an error code of 1 always indicates an end-of-file condition, and an error code of 15 indicates that a particular file does not exist. Many of Prime's C routines employ the same set of error codes as a convenience to C programmers. See the *Subroutines Reference Guide* for more information on error codes.

**File System Object:** An organized collection of data stored on a disk or some other storage medium (such as magnetic tape). File system objects include files, directories, and access categories.

**Fullword:** 32 bits (2 halfwords) or 4 bytes.

**Halfword:** 16 bits or 2 bytes.

**Master File Directory (MFD):** A special directory that contains the names of all *User File Directories (UFDs)* on a particular disk or partition. In PRIMOS, there is one MFD for each logical disk.

**On-unit:** A user-written routine that is coded to handle certain error conditions that may cause a program to abort.

**Origin Directory:** The directory to which you are attached when you log in to PRIMOS. This directory is also commonly referred to as your *initial attach point*.

**Pathname:** A multi-part name that specifies a particular PRIMOS file system object. A full pathname consists of the names of a disk volume, a UFD, a chain of subdirectories, and a target file system object.
Pass by Reference: Arguments are passed by reference from one function to another function when the address of the arguments in the calling function is passed to the called function. Consequently, the called function may modify the arguments in the calling function.

Pass by Value: Arguments are passed by value from one function to another function when a copy of the calling function's arguments is passed to the called function. Consequently, the called function cannot modify the arguments in the calling function.

PRIMOS: Prime Computer's operating system.

SEG Utility: SEG is the utility used to load and execute V-mode programs.

Source File: A file containing programming language statements in the format required by the appropriate compiler or assembler.

User File Directory: A directory listed in the MFD of a particular disk volume or partition.

V Mode: The addressing mode used for multi-segmented programs under PRIMOS.
Program Structure
A C program consists of one or several functions, which perform a specified task in a program. A function is equivalent to a subroutine in FORTRAN or a procedure in PASCAL. The C language does not impose a restriction on the number of functions that can be used in a single program. However, there must be at least one function named main. The main function is the point at which program execution begins. C functions must always be followed by a pair of parentheses, for example:

main ( )

Function Definitions
A function definition consists of two parts: a function header and a function body. The function header defines the name of the function and its arguments. The function body defines the purpose of the function, including any local argument declarations. C
functions may or may not return a value. A function
definition is formatted as follows:

[ type ] function-name( [ argument-list ] )
argument-declarations;
{
    local-declarations;
    statements;
}

Function Declarations

All C functions must be declared before they can be
used. A function declaration takes the format:

{ type } function-name( [ argument-list ] )
argument-declarations;

Identifiers

C language identifiers can be made up of digits, let-
ters, dollar signs ($), and underscore characters (_). However, the first character must not be a digit. On
Prime machines, identifiers can contain up to 32
characters. Use of the -NOCOMPATIBILITY option
results in identifiers being truncated to eight
characters.

Upper and lowercase identifiers yield different results.
For example, the identifiers step, STEP, and Step are
interpreted differently by the C compiler. Therefore, it is recommended that lowercase characters be used
to declare regular identifiers.
Comments

Comments are delimited by the characters /* and */, and can be placed anywhere in the source code where white space is present. The text of a comment can contain any combination of characters except the delimiter that is used to terminate a comment (*/). Comments can be one line long or they can span several lines.

Keywords

Keywords are predefined identifiers that cannot be redeclared. Keywords are used to identify the various data types, storage classes, and control-flow statements. Keywords must always be entered in lowercase letters and cannot be abbreviated.

<table>
<thead>
<tr>
<th>auto</th>
<th>extern</th>
<th>sizeof</th>
</tr>
</thead>
<tbody>
<tr>
<td>break</td>
<td>float</td>
<td>static</td>
</tr>
<tr>
<td>case</td>
<td>for</td>
<td>struct</td>
</tr>
<tr>
<td>char</td>
<td>fortran*</td>
<td>switch</td>
</tr>
<tr>
<td>continue</td>
<td>goto</td>
<td>typedef</td>
</tr>
<tr>
<td>default</td>
<td>if</td>
<td>union</td>
</tr>
<tr>
<td>do</td>
<td>int</td>
<td>unsigned</td>
</tr>
<tr>
<td>double</td>
<td>long</td>
<td>void*</td>
</tr>
<tr>
<td>else</td>
<td>register</td>
<td>while</td>
</tr>
<tr>
<td>entry</td>
<td>return</td>
<td></td>
</tr>
<tr>
<td>enum*</td>
<td>short</td>
<td></td>
</tr>
</tbody>
</table>

*Denotes Prime extensions to the C language.
Declarations

All variables must be declared before they can be used. A declaration consists of a storage class (optional), a data type, and a list of one or more variable names. For example:

```c
auto int lower, upper, step;
```

Variables can also be distributed among declarations, as follows:

```c
auto int lower;
auto int upper;
auto int step;
```

Data Types

There are four basic data types in the C language: `char`, `int`, `float`, and `double`. Additional data types can be derived by using the qualifiers `long`, `short`, and `unsigned`. The following declarations are valid:

```c
short int x;
long int x;
unsigned int x;
```

If you do not specify a data type in a declaration, the data type is assumed to be an integer.

Prime has added two additional data types to the C language: `enum` and `void`. The C User's Guide presents more information on the `enum` and `void` data types.

Storage Classes

Storage classes define the storage location and the lifetime of a variable. There are four storage classes in C: `auto`, `register`, `static`, and `extern`. 
Prime has added another storage class to the C language. The fortran storage class forces arguments to be passed by reference when calling procedures. The C User's Guide contains more information on the fortran storage class.

**Structures**

A structure is a group of related variables consisting of different data types, which are placed under a common identifier. The individual variables that comprise a structure are called members. The members of a structure can be the same type or a combination of different types. Structures are analogous to records in PASCAL. The following declaration is for a structure named car:

```c
struct car {
    int year;
    char color;
    char model;
};
```

**Unions**

A union is a variable that can store values of different types and sizes. Syntactically, a union declaration is similar to a structure declaration, except that the keyword struct is replaced by the keyword union. Below is an example of a union declaration.

```c
union mixed_array {
    int u_int;
    float u_float;
    char *u_ptr;
};
```
Arrays and Strings

An array is a collection of variables that are arranged in an ordered sequence. All arrays must be declared before they can be used. An array declaration takes the format:

```
type identifier[n];
```

A character string or string constant is an array of ASCII characters that are surrounded by double quotation marks (" "). Prime's C compiler uses the ASCII null character (\0) to terminate strings. The following is an example of a string.

"This is a C string!"

Pointers

A pointer is a variable that points to another variable, which contains a data value. Pointer declarations can be indicated as follows:

```
type *identifier;
```

The C Preprocessor

The C preprocessor is a simple text processor that handles file inclusion, macro substitution, and conditional text inclusion. All statements to the C preprocessor are preceded by a pound sign (#). The C preprocessor commands are:

```
#define     #ifdef     #if
#undef      #ifndef    #endif
#include    #else
```
Prime has added several preprocessor commands to the C language. These commands are \#list, \#nolist, and \#endincl. The \#list and \#nolist commands start and terminate output respectively to a specified file, and the \#endincl command logically terminates all \#include files prior to their physical end.
The precedence and order of evaluation of C operators are presented below. The term associativity refers to the method in which operators having equal precedence are evaluated.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary: ( ) [ ] -&gt; .</td>
<td>left to right</td>
</tr>
<tr>
<td>unary: ! ~ + + -- (type) * &amp; sizeof</td>
<td>right to left</td>
</tr>
<tr>
<td>multiplicative: * / %</td>
<td>left to right</td>
</tr>
<tr>
<td>additive: + -</td>
<td>left to right</td>
</tr>
<tr>
<td>shift: &lt;&lt; &gt;&gt;</td>
<td>left to right</td>
</tr>
<tr>
<td>relational: &lt; &lt;= &gt; =</td>
<td>left to right</td>
</tr>
<tr>
<td>equality: = = !=</td>
<td>left to right</td>
</tr>
<tr>
<td>bitwise: &amp;</td>
<td>left to right</td>
</tr>
<tr>
<td>bitwise:</td>
<td></td>
</tr>
<tr>
<td>logical: &amp;&amp;</td>
<td>left to right</td>
</tr>
<tr>
<td>logical:</td>
<td></td>
</tr>
<tr>
<td>conditional: ?:</td>
<td>right to left</td>
</tr>
<tr>
<td>assignment: = += -= etc.</td>
<td>right to left</td>
</tr>
<tr>
<td>comma: ,</td>
<td>left to right</td>
</tr>
<tr>
<td>bitwise: ^</td>
<td>left to right</td>
</tr>
</tbody>
</table>
This section provides instructions for invoking the C compiler, loading C programs, and executing loaded C programs.

**Invoking the C Compiler**

To invoke the C compiler, type the following command:

```
OK, CC SOURCEFILE [-option 1] [-option 2] ... [-option n]
```

**Compiler Options**

C compiler options come in pairs. That is, for each option there is an option having the opposite effect. One option of each pair is always the default. Each default option is indicated by an asterisk (*) and rust-colored letters indicate valid abbreviations. Compiler options can be specified in any order.
<table>
<thead>
<tr>
<th>Option</th>
<th>Operation Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>-64V*</td>
<td>Generate 64V mode code format.</td>
</tr>
<tr>
<td>-BIG</td>
<td>Assume external arrays span segments.</td>
</tr>
<tr>
<td>-BINARY* [pathname]</td>
<td>Generate binary (object) file. This option may take an argument. The default binary file is FILENAME.BIN.</td>
</tr>
<tr>
<td>-BIT8*</td>
<td>Set bit 8 on character constants.</td>
</tr>
<tr>
<td>-CHECKOUT</td>
<td>Execute the compiler's first pass only.</td>
</tr>
<tr>
<td>-COMPATIBILITY</td>
<td>Compile Version 6 source code as well as Version 7 and System III source code.</td>
</tr>
<tr>
<td>-CONVERT*</td>
<td>Convert parameter type to long.</td>
</tr>
<tr>
<td>-COPY*</td>
<td>Pass parameters by value.</td>
</tr>
<tr>
<td>-DEBUG</td>
<td>Generate information for full Source Level Debugger (DBG) support.</td>
</tr>
<tr>
<td>-DEFINE name [value]</td>
<td>Define a specified name to be a value.</td>
</tr>
<tr>
<td>-ERRTTY</td>
<td>Display error messages at user terminal.</td>
</tr>
<tr>
<td>-EXPLIST</td>
<td>Generate an expanded listing file.</td>
</tr>
<tr>
<td>-FRN</td>
<td>Generate floating point round number instruction.</td>
</tr>
<tr>
<td>Option</td>
<td>Operation Performed</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>-INCLUDE pathname</code></td>
<td>Specify include search pathnames.</td>
</tr>
<tr>
<td><code>-INPUT pathname</code></td>
<td>Designate the source file to be compiled. You must specify a pathname. This option is obsolete and its use is not recommended.</td>
</tr>
<tr>
<td><code>-INTL*</code></td>
<td>Generate 4-byte integers.</td>
</tr>
<tr>
<td><code>-INTS</code></td>
<td>Generate 2-byte integers.</td>
</tr>
<tr>
<td><code>-LISTING [pathname]</code></td>
<td>Generate a listing file. This option may take an argument. The default listing file is <code>FILENAME.LIST</code>.</td>
</tr>
<tr>
<td><code>-NOBIG*</code></td>
<td>External arrays do not span segments. This option uses 16-bit addressing.</td>
</tr>
<tr>
<td><code>-NOBIT8</code></td>
<td>Do not set bit 8 on character constants.</td>
</tr>
<tr>
<td><code>-NOCOMPATIBILITY*</code></td>
<td>Do not compile Version 6 source code.</td>
</tr>
<tr>
<td><code>-NOCONVERT</code></td>
<td>Do not convert parameter types.</td>
</tr>
<tr>
<td><code>-NOCOPY</code></td>
<td>Pass parameters by reference.</td>
</tr>
<tr>
<td><code>-NOERRTTY</code></td>
<td>Do not display error messages at user terminal.</td>
</tr>
<tr>
<td><code>-NOEXPLIST*</code></td>
<td>Do not generate an expanded listing file.</td>
</tr>
<tr>
<td><code>-NOFRN*</code></td>
<td>Do not generate FRN instructions.</td>
</tr>
<tr>
<td>Option</td>
<td>Operation Performed</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-NOPOP*</td>
<td>Remove old constant macro definition.</td>
</tr>
<tr>
<td>-NOSILENT*</td>
<td>Display severity-one error messages at user terminal.</td>
</tr>
<tr>
<td>-NOSTATISTICS*</td>
<td>Do not generate compiler statistics.</td>
</tr>
<tr>
<td>-NOVERBOSE*</td>
<td>Do not display severity-zero error messages at user terminal.</td>
</tr>
<tr>
<td>-POP</td>
<td>Push old constant macro definition onto a stack.</td>
</tr>
<tr>
<td>-PRODUCTION</td>
<td>Generate information for partial DBG support.</td>
</tr>
<tr>
<td>-SILENT</td>
<td>Do not display severity-one error messages at user terminal.</td>
</tr>
<tr>
<td>-SOURCE pathname</td>
<td>This option is identical to the -INPUT option. The -SOURCE option is obsolete and its use is not recommended.</td>
</tr>
<tr>
<td>-STATISTICS</td>
<td>Generate compiler statistics at user terminal.</td>
</tr>
<tr>
<td>-VERBOSE</td>
<td>Display severity-zero error messages at user terminal.</td>
</tr>
<tr>
<td>-XREF*</td>
<td>Generate a full cross reference listing.</td>
</tr>
<tr>
<td>-XREFS</td>
<td>Generate a partial cross reference listing of symbols referenced at least once after they have been declared.</td>
</tr>
</tbody>
</table>
Loading C Programs With SEG

After compiling a C program, you can use the SEG utility and its VLOAD subprocessor to produce an executable file with the suffix .SEG. A typical loading session using SEG is as follows:

```
OK; SEG-LOAD /*Invoke SEG utility*/
{SEG rev 19.3.3}
$ LIBRARY CCMAIN /*Load run-time library CCMAIN*/
$ LOAD PROGRAM /*Load main program's object file*/
$ LIBRARY CCLIB /*Load language-specific library*/
$ LIBRARY /*Load system libraries*/
LOAD COMPLETE /*Loader indicates load is complete*/
$ QUIT /*Save executable file and return to PRIMOS command level*/
OK,
```

If SEG does not display the message LOAD COMPLETE at the end of the loading procedure, you can use the MAP3 subcommand to check for any unresolved subroutine, program, or common block references. The MAP3 subcommand is described in the SEG and LOAD Reference Guide (DOC3524-192).
Executing C Programs

After you have loaded your C program, you can execute it by issuing the SEG command followed by a carriage return. For example, an executable file named SAMPLE.SEG would be executed as follows:

OK > SEG SAMPLE  /*.SEG suffix can be omitted*/

Executing Programs Within the Subprocessor

Upon receiving the LOAD COMPLETE message, you can issue the EXECUTE command to save the loaded program and begin immediate execution without first having to terminate SEG operation. When the C program has completed executing, SEG automatically returns to PRIMOS command level. The example below illustrates the use of the EXECUTE command.

LOAD COMPLETE
* EXECUTE
OK.

Compiling, Loading, and Executing With Command Files

You can save valuable time in compiling, loading, and executing C programs by creating a Command Procedure Language (CPL) file that will automatically perform these tasks for you. To find out more about CPL files, see the Prime User’s Guide and the CPL User’s Guide (DOC4302-190).
This section presents the library functions supplied with Prime's C compiler. The C library functions are stored in the runtime library CCLIB. The C User's Guide describes in detail each C library function presented in this section.

Most of the C library functions require the use of constants and keys in the calling sequence. These constants and keys have been defined in a set of header files, which are contained in the UFD designated SYSCOM.

<table>
<thead>
<tr>
<th>Header File</th>
<th>Library Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctype.h</td>
<td>Used by character classification functions</td>
</tr>
<tr>
<td>math.h</td>
<td>Used by mathematical functions</td>
</tr>
<tr>
<td>setjmp.h</td>
<td>Used by setjmp and longjmp functions</td>
</tr>
<tr>
<td>stat.h</td>
<td>Used by stat and fstat functions</td>
</tr>
<tr>
<td>stdio.h</td>
<td>Used by input and output functions</td>
</tr>
<tr>
<td>term.h</td>
<td>Used by gterm and sterm functions</td>
</tr>
<tr>
<td>time.h</td>
<td>Used by ctime and localtime functions</td>
</tr>
<tr>
<td>timeb.h</td>
<td>Used by ft ime function</td>
</tr>
</tbody>
</table>
Interpreting Definitions of Functions

This section uses a format for showing the header file required by each function, the function's parameter list, the parameter type the function uses, and the type of value the function returns. For example, the format used for the ftell function is:

```c
#include <stdio.h>
int ftell(file_pointer)
FILE *file_pointer;
```

The above definition for the ftell function indicates the following:

- The ftell function requires the use of the stdio.h header file.
- The ftell function returns an integer value.
- The ftell function has only one parameter, file_pointer, which is a pointer to FILE (a named type defined in the stdio.h header file).

Note
Although the example above resembles actual C code, it is not to be used as an example for defining C functions.

The C library functions are presented below in alphabetical order.

► abort

Raises the PRIMOS ABORT$ condition and immediately terminates program operation.

**DEFINITION**

`abort()`
abs

Returns the absolute value of an integer.

**DEFINITION**

```c
int abs(integer)
int integer;
```

access

Examines a specified PRIMOS pathname to determine whether the specified access rights are allowed. The following modes are valid:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Test to see if the file exists.</td>
</tr>
<tr>
<td>2</td>
<td>Write access</td>
</tr>
<tr>
<td>4</td>
<td>Read access</td>
</tr>
</tbody>
</table>

**DEFINITION**

```c
#include <stdio.h>
int access(pathname,mode)
char *pathname;
int mode;
```

You can specify more than one access mode by combining the values. For example, the value 6 indicates that the file exists and has read and write access.
 acos

Returns a value in the range of 0 to pi, which is the arc cosine of the radian argument.

DEFINITION

#include <math.h>
double acos(x)
double x;

 asin

Returns a value in the range -pi/2 to pi/2, which is the arc sine of the radian argument.

DEFINITION

#include <math.h>
double asin(x)
double x;

 atan

Returns a value in the range -pi/2 to pi/2, which is the arc tangent of the radian argument.

DEFINITION

#include <math.h>
double atan(x)
double x;
► atan2

Returns a value in the range -pi to pi. The returned value is the arc tangent of x/y, where x and y are two arguments.

DEFINITION

#include <math.h>
double atan2(x,y)
double x,y;

► atof

Converts an ASCII character string to its appropriate numeric value.

DEFINITION

#include <math.h>
double atof(nptr)
char *nptr;

► atoi

Converts an ASCII character string to its appropriate numeric value.

DEFINITION

#include <math.h>
int atoi(nptr)
char *nptr;
**atol**

Converts an ASCII character string to its appropriate numeric value.

**DEFINITION**

```c
#include <math.h>
Long atol(nptr)
char *nptr;
```

**cabs**

Returns $\sqrt{x^2 + y^2}$.

**DEFINITION**

```c
#include <math.h>
double cabs(z)
struct{double x,y;} z;
```

**calloc**

Allocates an area in memory.

**DEFINITION**

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

**ceil**

Returns as a **double** value the smallest integer that is equal to or greater than its argument.

**DEFINITION**

```c
#include <math.h>
double ceil(x)
double x;
```
- cfree

Frees a previously allocated area in memory.

**DEFINITION**

```c
int cfree(pointer)
char *pointer
```

- chdir

Changes the current working directory to a specified PRIMOS pathname.

**DEFINITION**

```c
#include <stdio.h>
int chdir(pathname)
char *pathname;
```

- chrcheck

Returns a value of 1 if you typed a character that has not yet been read. A value of 0 is returned if there is no such character.

**DEFINITION**

```c
int chrcheck()
```