**Reading a Directory**

* A directory is really a special type of file, a data container that acts as a database referencing other files stored on the media.
* The media’s file system determines how the files are organized and accessed.
* The directory holds all that information, such as the file’s physical location, its name, timestamps, permissions, and other trivia. These details are accessible when you use the proper C language functions.

To access a directory, use the *opendir()* function. It’s prototyped in the dirent.h header file as:

DIR \*opendir(const char \*filename);

The function requires a string argument, a name or path to a directory. The value returned is a *DIR* pointer, similar to the *FILE* pointer returned by *fopen()*.

After opening the directory and doing whatever, you use the *closedir()* function to close the directory:

int closedir(DIR \*dirp);

The function requires a *DIR* pointer (dirp) and returns 0 upon success, otherwise -1.

Use the [errno](https://c-for-dummies.com/blog/?p=1735) global variable to further examine the issue when -1 is returned.

In the following code, the current directory (abbreviated as . ) is opened and closed:

#include <dirent.h>

int main()

{

DIR \*folder;

folder = opendir(".");

if(folder == NULL)

{

puts("Unable to read directory");

return(1);

}

else

{

puts("Directory is opened!");

}

closedir(folder);

return(0);

}

The *DIR* pointer variable folder acts as the directory handle for the *opendir()* function in Line 8. If it’s NULL, the program quits. Otherwise a simple string is output and the directory is closed in Line 18. Here’s sample output:

Directory is opened!

To expand the program and make it useful, the *readdir()* function is added. Here’s how it’s prototyped, also in the dirent.h header file:

struct dirent \*readdir(DIR \*dirp);

Structure variable dirent is a pointer that contains information about a specific entry read from a directory, referenced by the *DIR* pointer used in the *readdir()* function.

This function is called repeatedly until NULL is returned, indicating no further entries are available in the directory.

The dirent structure is customized based on the file system. A key member worthy of access is d\_name, which represent the directory entry’s filename. The following code demonstrates how the d\_name structure member can be used when reading a directory:

#include <stdio.h>

#include <dirent.h>

int main()

{

DIR \*folder;

struct dirent \*entry;

int files = 0;

folder = opendir("E:\Winter\_2020-21\ACP\Mod\_5 ACP");

{

perror("Unable to read directory");

return(1);

}

while( (entry=readdir(folder)) )

{

files++;

printf("File %3d: %s\n",

files,

entry->d\_name

);

}

closedir(folder);

return(0);

}

This code is the same as the previous example, with a *while* loop inserted to read the directory entries. The loop spins as long as the value returned from *readdir()* isn’t NULL. For each item found, *int* variable files tracks its sequential number (which is part of this code and not part of the file system) and entry->d\_name prints its name:

File 1: .

File 2: ..

File 3: 0825a.c

File 4: 0825b.c

File 5: 08exercise-a.c

File 6: 08exercise-b.c

File 7: 0901.c

File 8: 0908.c

File 9: a.out

File 10: gettysburg.txt

File 11: sto

Directory entries . and .. represent the current and parent directories.

**Directory Entry File Types**

Not every file listed in a directory is the same. Some entries represent other directories, subdirectories, for example. To determine which entries are mere mortal files and which are directories, you must examine the file type.  
  
File type information is held in the same dirent structure that yields the file’s name. The structure member is d\_type.

The following code modifies the previous example to display the d\_type member’s value for the current directory:

#include <stdio.h>

#include <dirent.h>

int main()

{

DIR \*folder;

struct dirent \*entry;

int files = 0;

folder = opendir(".");

if(folder == NULL)

{

perror("Unable to read directory");

return(1);

}

while( (entry=readdir(folder)) )

{

files++;

printf("File %3d: %s, %d\n",

files,

entry->d\_name,

entry->d\_type

);

}

closedir(folder);

return(0);

}

The only modification to this code from is the *printf()* statement starting at Line 20, the entry->d\_type item and used the %d placeholder to display its integer value. Here’s sample output:

File 1: .,4

File 2: ..,4

File 3: 0825a.c,8

File 4: 0825b.c,8

File 5: 08exercise-a.c,8

File 6: 08exercise-b.c,8

File 7: 0901.c,8

File 8: 0908.c,8

File 9: a.out,8

File 10: gettysburg.txt,8

File 11: sto,4

The values returned are specific to the operating system. Under Unix, value 8 is a regular file and 4 is a directory. On my PC, the MinGW compiler’s C library lacks the dirent->d\_type member. So the code doesn’t compile. That’s okay, because a better way is available to read whether a directory entry is a file or directory: Use the *stat()* function, as defined in the sys/stat.h header file:

int stat(const char \*restrict path, struct stat \*restrict buf);

//get file status information

The *stat()* function requires two arguments. The first is the name (or pathname) to a filename. The second argument is the address of a stat structure. This structure is filled with oodles of good info about a directory entry and it’s consistent across all file systems.

The sys/stat.h header file also defines macros to test for file type, which work similarly to the ctype.h macros that examine characters. For a directory entry, the S\_ISDIR macro is used:

#include <stdio.h>

#include <dirent.h>

#include <sys/stat.h>

int main()

{

DIR \*folder;

struct dirent \*entry;

struct stat filestat;

folder = opendir(".");

if(folder == NULL)

{

perror("Unable to read directory");

return(1);

}

/\* Read directory entries \*/

while( (entry=readdir(folder)) )

{

stat(entry->d\_name,&filestat);

if( S\_ISDIR(filestat.st\_mode) )

printf("%4s: %s\n","Dir",entry->d\_name);

else

printf("%4s: %s\n","File",entry->d\_name);

}

closedir(folder);

return(0);

}

The *stat()* function appears at Line 21. Its arguments are the name of the file located in the directory (entry->d\_name) and the address of the stat structure filestat, declared at Line 9.

In Line 22, the S\_ISDIR macro tests the filestat.st\_mode member. If true, the entry is a directory and it’s flagged as such in the output. Otherwise, the *else* statement flags the entry as a standard file.

Here’s sample output:

Dir: .

Dir: ..

File: 0825a.c

File: 0825b.c

File: 08exercise-a.c

File: 08exercise-b.c

File: 0901a.c

File: 0901b.c

File: 0908.c

File: a.out

File: gettysburg.txt

Dir: sto

This code runs on both the PC and Unix systems, thanks to the *stat()* function.

You can pull more details from the *stat()* function, including file size and timestamps. I’ll cover that process in next week’s Lesson.

This entry was posted in [**Lesson**](https://c-for-dummies.com/blog/?cat=2) by **[dgookin](https://c-for-dummies.com/blog/?author=1)**. Bookmark the [**permalink**](https://c-for-dummies.com/blog/?p=3252).