Reading Directories

The program will use library calls to open the given Linux directory (or the current directory if no command line argument is given) for reading. It will read every directory entry, printing out each file's inode number, numerical type, symbolic type and name.

Note that the Macintosh implementation of `dirent` is different from that of Linux.

Samples

Output when no command line argument is given

```
% ./a.out
11468143             0x04 D .
1833880              0x04 D ..
11468217             0x08 O README.md
51723738             0x08 O a.out
11468218             0x08 O main.c
11468214             0x08 O .gitignore
11468220             0x08 O project.s
11468144             0x04 D .git
11468215             0x04 D .vscode
%
```

The first column is the named file's inode number. Think of an inode number as a file's unique (per file system) serial number. You must print it left justified in a field of 20 digits using `printf`.

Since it is possible you've never used `printf` before, I will supply the correct template string in the C version of the program presented below.

The second column is the file's type printed as a single byte's worth of hex.

The third column is a single character chosen to symbolically match the value in the second column.

The fourth column is the file's name.

Output when a command line argument is given

```
% ./a.out /usr/share/vim
1152921500312447599  0x04 D .
1152921500312429992  0x04 D ..
1152921500312447600  0x04 D vim90
1152921500312451414  0x08 O vimrc
%
```

It says that `.`, `..` and `vim90` are directories and that `vimrc` is an ordinary file.

Output when a bad command line argument is given

```
% ./a.out nibble
nibble: No such file or directory
%
```

The error string is produced by `perror()`.

man

Here is where you will get documentation for `perror()`, `opendir()`, `closedir()`, and `readdir()`. The man page for `readdir()` also describes `struct dirent`.

```
man perror
man opendir
man closedir
man readdir
```

`man` is your friend, though of course in the 21st century it should be called `person`. To learn more about `man`, do the obvious thing:

```
man man
```

FOR THIS PROJECT DON'T USE MAN FROM A MAC TERMINAL – WHY? STEVE FRIGGEN JOBS THAT'S WHY.

It will be equally pointless to try the above Linux shell commands from a Windows command prompt but hey - give it a try.

The reason to not read the `man` pages from the Mac is that everything beyond the name of the functions you need will be different. You know, "Think Different."

`opendir()`

This function takes a NULL terminated `C-string` and attempts to open it as a directory. Get the details from the `man` page. If you get an error return, pass the attempted directory name to `perror()` to get the right error message.

`closedir()`

Call this function to close a successfully opened directory. Get the details from the `man` page.

`readdir()`

Call this function to be given a pointer to the next `dirent` or `NULL` if there are no more (or there is an error). Pay attention to the `man` page to distinguish between no more `dirent` structures and an error. In short, `errno` should be initialized to 0 then checked once you've gotten a `NULL` back from `readdir()`.

Source code to a C version

Here is the source code to my C version.

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

int main(int argc, char ** argv) {
    int retval = 1;
    char * dirname = ".";
    char *fmt = "%-10llu 0x%02x %c %s\n";

    if (argc > 1)
        dirname = argv[1];

    DIR * dir = opendir(dirname);
    if (dir) {
        struct dirent * de;
        errno = 0;
        while ((de = readdir(dir)) != NULL) {
            char symbol;
            switch (de->d_type) {

            case DT_CHR:
                symbol = 'C';
                break;

            case DT_DIR:
                symbol = 'D';
                break;

            case DT_FIFO:
                symbol = 'P';
                break;

            case DT_LNK:
                symbol = 'L';
                break;

            case DT_REG:
                symbol = 'O';
                break;

            case DT_SOCK:
                symbol = 'S';
                break;

            default:
                symbol = '?';
                break;
            }
            printf(fmt, de->d_ino, de->d_type, symbol, de->d_name);
        }
        if (errno != 0)
            perror("readdir() failed");
        closedir(dir);
        retval = (errno != 0); // force error return to be 1
    }
    else
        perror(dirname);
    return retval;
}
```

Getting the address of `errno`

`errno` is an `extern`. To store anything into it (or query its contents), you must have its address. For reasons which will not be explained, getting its address is accomplished by calling a library function.

Remember to properly set the return value of `main()`

If all ends well, zero should be returned from `main()`. If any error is found, a value of 1 should be returned.

Check in this way:

```
echo $?
```

`$?` is a shell variable that contains the value returned from the last program run by the shell.

Setting the symbol to its correct value

The above source code indicates the valid symbols as various cases of the `switch` statement. You can implement the `switch` as a series of if statements (in assembly language, of course) or you can do it in a relatively tiny number of statements using an array or rather, a C-string 13 bytes long (not including a null terminator).

My specifying the exact length of the C-string is in fact, a strong hint as to how to go about this.

Those who correctly make use of the C-string method will be awarded five points above their project grade (i.e. extra credit).

Likely source of error

If you're printing garbage, double check your calculations of offsets within the `dirent`. While this isn't the only explanation, it is a likely explanation.

What to turn in

Turn in only your nicely commented assembly language program.

Work Rules

All work is to be done solo.