Programming Project: ls-R1fF

Introduction

For this assignment you will write a program that implements some of the options of the Unix **ls** command. In doing so, you will have a chance to work with the Unix directory structure, file types, recursion, and, optionally, the stat() system call.

Explanation of ls -R1fF

Imagine you have set up a complex, multi-level directory structure to organize your files. Directories contain subdirectories which in turn contain subdirectories of their own. How can you get a list of all the contents of this tree of directories and files?

The **ls** commmand lists the contents of directories. It can also tell you information about the files, such a the owner, size, and modification time. **ls** supports many command-line options. If you want to list all the files and directories in and below your home directory, you can type:

```
$ ls -R1fF $HOME
```

The **ls** command lists information about the directories or files named on the command line. The command line options tell ls what information to report and how to report it. Read the manual page on **ls** for more info.

Explanation of lsr1ff

For this assignment, you will write a program that operates exactly like **ls** with the **-R1fF** options. The syntax of your program (to be called **lsr1ff**, because typing mIxeD caSe is annoying), is:

```
$ lsr1ff [file_or_dir ..]
```

for example, the command

```
$ lsr1ff /etc /bin/who /bin/whp
```

lists recursively all files and directories under the /etc directory, displays the name /bin/who with an asterisk appended, and reports that /bin/whp does not exist.

Getting Started

1. Read the manual page on the **ls** command. Try it out in your own directory or in the course directory. For example, you might try:

```
$ ls -R1fF ~COURSE/lectures/lect03
```

2. cd to ~COURSE/hw/lsr1ff and explore the directory trees under tdir1 and tdir2 . See what sort of files and directories are in that tree. Try out ls with various combinations of the options **R**, **1**, **f**, and **F**. Make sure you understand what each option does.

3. Once you see how real ls works and have made sense of each of the options, think about what your ls has to do. If there are no arguments, it lists the tree starting at the current directory. If there are arguments, your program has to process each argument in turn. If an argument names a file, your program displays the name with the attribute symbol appended. If the argument is a directory (use stat(), lstat(), or isadir()) your program has to process all the items named in the directory. Therefore the search and print function will be

recursive.

4. Copy the file ~COURSE/hw/lsr1ff/isadir.c to your account. This simple file provides a function you can use to test if something is a directory.

5. Study the versions of ls from the text. ls2.c shows how to loop through command line arguments, how to loop through the contents of a directory, and how to implement the **-l** option. You can start from scratch, or you can rework this sample code.

You might find it useful to write a function called listdir(char *dirname). This function lists the contents of a directory with attribute characters appended. In addition, if any entry in the directory is a directory, then listdir() will have to create a new string to hold the new directory name and pass that string to itself. Use malloc() to create the string; a fixed size buffer is liable to run out when you least expect it. Examine the output of **ls -R1fF** to see the format and sequence of items it prints.

Building Your Program

Write your own file or files of code. Compile isadir.c to isadir.o by typing `cc -c isadir.c` . Then link your files with isadir.o to make lsr1ff. (command looks like: `cc lsr1ff.c isadir.o -o lsr1ff`).

Testing Your Program

You can make up your own test system data, but you must test it sometime with ~COURSE/hw/lsr1ff/test.lsr1ff . This shell script will exercise your program on a preset directory tree. To run it, type:

```
$ ~COURSE/hw/lsr1ff/test.r1ff
```

Things to Worry About

There is no limit on the depth of a directory tree; any directory can contain subdirectories. An example in the text class shows how to create extremely deep structures.

On the other hand, there is a limit to the number of files and directories a process may have open at one time. That number was 20 in ancient versions of Unix, and now that limit is 60 or more.

How will your version of ls respond when it encounters a very deep tree?

Turn in Your Work

Hand in (1) a copy of the source to your program, (2) a sample of its output from the test.lsr1ff run mentioned above, and any other examples you want to include, and (3) a short paragraph (not more than six sentences) answering the question in 'Things to Worry About' mentioned above.