Traverse a Directory

Write TWO C programs. One use recursion and one does not use recursion. Each program sorts all the files in a directory based on file sizes, including all the files under all the sub-directories, sub-sub-directories, etc. (all the levels).

Submission Instructions

Submit individual files following the instructions below. DO NOT SUBMIT A ZIP FILE. NJITID# is the eight-digit NJIT ID (Not your UCID, Rutgers students also have NJIT IDs). Index is the index number showing the order of the screenshots (e.g., 1 or 2).

- **your C programs**: Name your program using the pattern NJITID#_resursive.c and NJITID#_nonresursive.c
- **1~2 screenshots in jpg format for each program showing that your program really works**: Name your screenshots using the pattern NJITID#_recursive_index.jpg or NJITID#_nonrecursive_index.jpg
- **1 screenshot in jpg format to show the difference between the two programs**. On the screenshot, **Highlight or circle the code that does the recursion**. You may use `diff` to get the differences and then take a screenshot. Name your screenshots using the pattern NJITID#_diff.jpg

  ```
  diff   NJITID#_resursive.c   NJITID#_nonresursive.c
  ```

Objectives

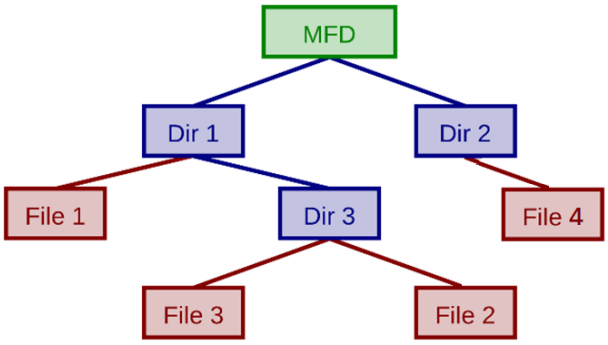
- To learn how to traverse a directory
- To learn how to check file status
- To gain more experience on using pointers and linked lists
- To gain more experience on sorting a linked list.
- To learn how to write recursive functions

Requirements and Instructions

- Your program should take one argument, which is the pathname of the directory. (Important! If you don't follow strictly, the grader may not be able to run your program correctly, and you may get lower grades.) For example, the following command sorts the files under directory `/usr/share/man`:

  ```
  ./your_program  /usr/share/man
  ```

- The files under the sub-directories at all levels (i.e., sub-directories, sub-sub-directories, sub-sub-sub-directories, etc) should also be considered and sorted. For example, when your program is run against directory *MFD* on the right, all the files, including *File 1*, *File 2*, *File 3*, and *File 4*, should be considered.



- For an implementation using recursion, refer to the *monitor* example in the slides for how to traverse a directory tree in a C program, particularly the code in *MonitorFile* and *processDirectory* functions. For an implementation without using recursion, use a linked list to organize the sub-directories that have not been scanned. Refer to the slides on directory traversal in the bash part. The slides on state space search can also give you some ideas.
- To simplify the problem, the program only needs to consider regular files (i.e., those with the `S_ISREG()` test returning 1). Also, in the problem, we assume that each regular file only has one hard link. So, the program does not need to look at the number of hard links when checking files.
- Symbolic links are links, and are not considered as regular files. Thus, when your program checks the status of a file, make sure that it calls `lstat()`, not `stat()`. Check the slides for the differences between `lstat()` and `stat()`.
- Assume the longest pathname does not exceed 256 single-byte characters.
- For the files with the same size, there is no requirement on how their pathnames should be sorted.
- Print out the results on the screen in text format, one line for each file with file size followed by a tabular symbol ('\t') and the file pathname. Programs using different formats in outputs will fail the tests and lead to lower grades.
  The sample output when the program is run with the command below against directory /usr/share/man/ is shown as follows:

```
$./your_program  /usr/share/man
30      /usr/share/man/man1/cpp-4.8.1.gz
30      /usr/share/man/man1/cpp-5.1.gz
30      /usr/share/man/man1/gcc-4.8.1.gz
30      /usr/share/man/man1/gcc-5.1.gz
35      /usr/share/man/man3/XauDisposeAuth.3.gz
35      /usr/share/man/man3/XauFileName.3.gz
35      /usr/share/man/man3/XauGetAuthByAddr.3.gz
…
86618   /usr/share/man/man1/bash.1.gz
104755  /usr/share/man/man5/smb.conf.5.gz
287622  /usr/share/man/man1/x86_64-linux-gnu-gcc-6.1.gz
303306  /usr/share/man/man1/x86_64-linux-gnu-g++-7.1.gz
303306  /usr/share/man/man1/x86_64-linux-gnu-gcc-7.1.gz
```

Testing: For each program, perform the following tests:

Test 1. The program can print out correct result when it is run against a directory containing a few regular files with different sizes

Step 1. Create a directory and a few files in the directory

```
mkdir ./test1
dd if=/dev/zero of=./test1/file1 bs=10 count=1
dd if=/dev/zero of=./test1/file2 bs=100 count=1
dd if=/dev/zero of=./test1/file3 bs=1000 count=1
```

Step 2. Execute the program against `./test1`:

```
./your_program   ./test1
```

Step 3. Check the output of the program, which should look like

```
10      ./test1/file1
100     ./test1/file2
1000    ./test1/file3
```

Test 2. The program can print out correct result when it is run against a directory containing sub-directories and regular files in sub-directories

Step 1. Create a directory and a few files in the directory

```
mkdir ./test2
dd if=/dev/zero of=./test2/file1 bs=10 count=1
dd if=/dev/zero of=./test2/file2 bs=100 count=1
mkdir ./test2/dir1
dd if=/dev/zero of=./test2/dir1/file3 bs=20 count=1
dd if=/dev/zero of=./test2/dir1/file4 bs=200 count=1
mkdir ./test2/dir2
dd if=/dev/zero of=./test2/dir2/file5 bs=30 count=1
dd if=/dev/zero of=./test2/dir2/file6 bs=300 count=1
mkdir ./test2/dir3
dd if=/dev/zero of=./test2/dir3/file7 bs=40 count=1
dd if=/dev/zero of=./test2/dir3/file8 bs=400 count=1
```

Step 2. Execute the program against `./test2`:

```
./your_program   ./test2
```

Step 3. Check the output of the program, which should look like

```
10      ./test1/file1
20      ./test1/dir1/file3
30      ./test1/dir2/file5
40      ./test1/dir3/file7
100     ./test1/file2
200     ./test1/dir1/file4
300     ./test1/dir2/file6
400     ./test1/dir3/file8
```

Test 3. The program can print out correct result when it is run against directory /usr/share/man.

Step 1: Run the program and redirect the result into a file *output_test.txt*

```
./your_program /usr/share/man > output_test.txt
```

Step 2: Check whether the sizes are correctly sorted in the output. If they are correctly sorted, you will not see *diff* print out any text.

```
cut -f 1 output_test.txt > sizes_test.txt
sort -s -n sizes_test.txt > sizes_test_sorted.txt
diff sizes_test.txt sizes_test_sorted.txt
```

Step 3: Run the following command on one line and redirect the result into another file *output_standard.txt*

```
find /usr/share/man -type f -print0 | xargs -0r du -bl | sort
  -k 1,1n -k 2,2 > output_standard.txt
```

Step 4: This step checks whether the output includes the sizes of any files. Compare the sizes contained in *output_test.txt* and *output_standard.txt* using the following commands. You should not see *diff* print out any text if the sizes of all the files have been correctly collected in the program.

```
cut -f 1 output_standard.txt > sizes_standard.txt
diff sizes_test_sorted.txt sizes_standard.txt
```

Step 5: This step tests whether the output includes the pathnames of all the files. Compare the pathnames in *output_test.txt* and *output_standard.txt* using the following commands. You should not see diff print out any text if the pathnames of all the files have been correctly collected in the program.

```
cut -f 2 output_test.txt | sort > filelist_test.txt
cut -f 2 output_standard.txt | sort > filelist_standard.txt
diff filelist_test.txt filelist_standard.txt
```