CS 332/532 – 1G- Systems Programming HW 2

Objectives

1. To implement a search program in C program using system calls for files and directories.

Assignment Submission

The assignment submission in Canvas is mandatory. No late submissions or extensions will be accepted.

Submission Checklist:

- Upload a C source file (.c file) to Canvas as part of this assignment submission. Submissions through the Canvas "Comments" will not be accepted. The file should be named in this naming convention: yourblazerid_HW02.c
- Upload a README.md file which should include:
  - Instructions on how to compile your C source file into an executable.
  - How to run the executable program.
  - Any citation documentation.
  - A link to your GitHub repository.
- Upload a Make.mk file which should include:
  - The build instructions for your program
- Upload an Independent Completion Form.

Please do not upload executables or object files.

Rubric

Criteria	332 Pts	532 Pts
Uploaded Independent Completion Form	5 pts	
Uploaded .c file Uploaded .c file of code submission.	5 pts	
Uploaded .md readme file and .mk MAKE file. Uploaded .md file of code description, usage information, and any coding citations. Uploaded .mk file of code compilation instructions.	5 pts	
Provided working (shared) Github repository link Provided working (shared) Github repository link in the README file.	5 pts	
“Basic Functionality” Compiles & Does Not Error	50 pts	30 pts
“More Details” Command-line Compiles & Does Not Error	10 pts	20 pts
“Large Files” Command-line Compiles & Does Not Error	10 pts	
“Search String” Command-line Compiles & Does Not Error	10 pts	
“File Types” Command-line Compiles & Does Not Error	N/A	10 pts

Your code needs to be compiled on GitHub codespace, or you need to demo your code to TA.

HW Assignment #2

'*Find'* is a popular UNIX command that traverses a file hierarchy and performs various functions on each file in the hierarchy. The goal of this homework is to implement a program similar called ***'search'*** that supports the following functionality:

1. "Basic Functionality": The program should accept as an input (command-line argument) the directory name from where to start the file traversal and print the file hierarchy starting with that directory.
    - If the program is executed without any arguments, the program should print the file hierarchy starting with the current directory where the program is executed.
    - If there are no directories in the current directory only files are listed one per line.
    - The start directory should be included in the output as the first entry.
    - If there are other directories in the current directory, then the directory name is first displayed on a separate line and then the files in that directory are listed one-per-line with one-tab indentation.
    - The program should iterate until there are no further files or directories to print out.
    - If a file is a symbolic link then the program should display the symbolic link name and in parentheses the file name the link points to.
    - The program should support not only each of the options (below) separately but also any combination of these options, in any order provided. For example: -v, -L 1024, -s jpg 1, -v -L 1024, -v -s jpg 2, -L 1024 -s jpg 2, -v -L 1024 -s jpg 1, -v -s jpg 2 -L 1024.
    - If both -L and -s options are specified, then the program should list only those files that match both criteria. The order of the options should not matter.
2. The program should also support three command-line options:
    1. -v
       "More Details": This should list all files in the hierarchy and print the size (in bytes), permissions (symbolic or octal representation), and last access date & time next to the filename in parenthesis (format: "Jan 17 13:00:00 2024"). Print 0 for the size of a directory.
    2. -L <file size in bytes>
       "Large Files": This should list all files in the hierarchy with a file size greater than or equal to the value specified.
    3. -s <string pattern> <depth>
       "Search String": This should list all files in the hierarchy that satisfy the following conditions: 1) the file name contains the substring in the string pattern option, AND 2) the depth of the file relative to the starting directory of the traversal is less than or equal to the depth option. The starting directory itself has a depth of 0.

3. [**Graduate Students Only**] "File Types": The program should support a fourth command-line option:
    1. *-t f*
        - List regular files only (no directories, symbolic links, etc). Must still show indentations to indicate the transversal of directories. Must still be able to be combined with other arguments.
    2. *-t d*
        - List directories only ('directory' file types). Must still show indentations to indicate the transversal of directories. Must still be able to be combined with other arguments.

Guidelines and Hints

1. The program must use **function pointers** similar to Figure 4.22 in the textbook to implement the functionality described above. You can use the logic and structure from Figure 4.22 as the starting point to implement this program (make sure to go over the program in Figure 4.22 and understand all the steps performed). However, please note that your final program must compile and execute without any dependencies on the source code provided by the text book. You can find a simple example on how to use function pointers in the funcptr.c file.
2. You can use the ***getopt*** function to process the command-line options. See "*man 3 getopt*" for more details and an example on how to use *getopt* function.
3. You should use a **Makefile** to compile and build this project and make sure to submit the Makefile along with the rest of the source code.

Program Documentation and Testing

1. Use appropriate names for variables and functions.
2. Use a Makefile to compile your program.
3. Include meaningful comments to indicate various operations performed by the program.
4. Programs must include the following header information within comments:

```
/*
Name:
BlazerId:
Project #:
To compile: <instructions for compiling the program>
To run: <instructions to run the program>
*/
```

5. Test your program with the sample test cases provided as well as your own test cases.
6. You can include any comments you may have about testing and citations in the README.txt file.

Examples

Command	***Description***
./search	List all files in the current directory where the program is executed. Remember in Unix, directories are a type of file.
./search ../programs	List all files in the directory ../programs (relative to the current directory)
./search /CS332/programs	List all files in the directory /CS332/programs (absolute path)
./search -v ../programs	List all files in the directory ../programs along with the required attributes
./search -L 1024	List all files with size >= 1024 bytes in the current directory
./search -L 1024 ../programs	List all files with size >= 1024 bytes in the ../programs (relative to the current directory)
./search -s jpg 1	List all files in the current directory that have the substring “jpg” in their name with depth <=1 relative to current directory
./search -s jpg 1 -L 1024	List all files in the current directory that have the substring “jpg” in their name with depth <=1 relative to the current directory and size >= 1024
./search -L 1024 -s jpg 1	The same as “./search -s jpg 1 -L 1024”
./search -v -s jpg 1 -L 1024	Similar with “./search -s jpg 1 -L 1024”. Print the required attributes, not only the filenames.

Sample Input and Output:

If you have the following directory structure as shown by the output of "ls -R" command:

```
$ ls -R projects projects:
fread.c fwrite.c project1 project2 project3 project4 read.c write.c

projects/project1:
project1.docx README

projects/project2:
project2.docx README

projects/project3:
project3.docx README

projects/project4:
project4.docx README
```

Then the output of find without any argument should look like this:

```
projects
      fread.c
      fwrite.c
      project1
            README
            project1.docx
      project2
            project2.docx
            README
      project3
            project3.docx
            README
      project4
            project4.docx
            README
      read.c
      write.c
```

It is not necessary that the order of the files are exactly as shown above, but the overall structure should look similar to the output shown above. You can use the following tar file to create the directory structure: projects.tar. Download this file and extract the file using the command:

```
$ tar xvf projects.tar
```