CS 332/532 – 1G- Systems Programming

HW 2

Deadline: 09/24/2023 11:59pm

Objectives

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

Description

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 project is to implement a program similar called *search* that supports the following functionality:

1. The program should take the directory name from where to start the file traversal as a command-line argument and print the file hierarchy starting with the directory that is provided by the command-line argument.
2. 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.
3. 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.
4. 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.
5. The program should also support three command-line options:
    1. -S
       This should list all files in the hierarchy and print the size, permissions, and last access time next to the filename in parenthesis. Print 0 for the size of a directory.
    2. -s <file size in bytes>
       This should list all files in the hierarchy with file size less than or equal to the value specified.
    3. -f <string pattern> <depth>
       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.
6. The program should support not only each of these options separately but also any combination of these options. For example: -S, -s 1024, -f jpg 1, -S -s 1024, -S -f jpg 2, -s 1024 -f jpg 2, -S -s 1024 -f jpg 1, -S -f jpg 2 -s 1024.

7. If both -s and -f options are specified then the program should list only those files that match both criteria. The order of the options should not matter.
8. [**Graduate Students Only**] The program should support a fourth command-line option:
    1. *-t f* - list regular files only
    2. *-t d* - list directories only

Guidelines and Hints

1. The program must use function pointers similar to Figure 4.22 in the text book 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.
4. You should upload all the source code, Makefile, and a README.txt file to Canvas. Please do not upload any object files or executable files.

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 in the README.txt file.

Examples

Command	*Description*
./search	List all files in the current directory where the program is executed.
./search ../programs	List all files in the directory ../programs (relative to the current directory)
./search /home/UAB/puri/CS332/programs	List all files in the directory /home/UAB/puri/CS332/programs (absolute path)
./search -S ../programs	List all files in the directory ../programs along with the required attributes
./search -s 1024	List all files with size <= 1024 bytes in the current directory
./search -s 1024 ../programs	List all files with size <= 1024 bytes in the ../programs (relative to the current directory)
./search -f 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 -f jpg 1 -s 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 -s 1024 -f jpg 1	The same as “./search -f jpg 1 -s 1024”
./search -S -f jpg 1 -s 1024	Similar with “./search -f jpg 1 -s 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 -lR" 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
```

Submission Guidelines

• Use best software engineering practices to design and implement this homework. The next homework will extend the functionality provided by this program.

• Use functions and organize these functions in multiple files if necessary.

• Use a Makefile to compile and build the multiple files.

• Document you code and include instructions for compiling and executing the program in the README.txt file.

• Test your program and describe how you tested this program in the README.txt file.

• To submit this homework :

- upload all the source files and documentation to Canvas.

Description	**Points**
Implementation of *search* that lists the files and directories in the specified format when executed with no arguments or when the directory name is specified as an argument	30 points
Correct use of function pointers to implement the required functionality	20 points
Implementation of *search* that lists the files and directories in the specified format when executed with -S option	10 points
Implementation of *search* that lists the files and directories in the specified format when executed with -s option	10 points
Implementation of *search* that lists the files and directories in the specified format when executed with -f option	10 points
Implementation of *search* that lists the files and directories in the specified format when executed with multiple options (combination of -S, -s, and -f)	10 points
Use of Makefile, source code documentation (including README..txt)	10 points

Grading Rubrics