CSI 402 – Systems Programming

Programming Assignment I

Date given: Sept. 16, 2016 **Due date:** Sept. 26, 2016
Total grade for this assignment: 100 points **Weightage:** 5%

A. Purpose. A typical question users may have regarding files on the file system is how big they are. There are a number of programs that can answer this question, such as the `ls` program that lists information about files of any type, including directories. Sometimes users might want to find the size of all files (that match some filename pattern) in all subdirectories. For example, a user might want to find the size of all C or text files in the current directory and all subdirectories. It is possible to do this with the `find` program that searches for files in a directory hierarchy.

Your task is to develop a modified version of the `ls` command so that it can find the size of all files that match some filename pattern in all subdirectories. Your program should take as input filename patterns, and report the size of each file that matches these patterns. Obviously, you are not allowed to use the `find` or `ls` programs. Instead, you are asked to go through the files that match the given filename pattern, open the files in succession and read through their contents to determine their size.

The purpose of this assignment is to familiarize yourself with (i) C programming, (ii) splitting a program into multiple files (including header files), (iii) using make and makefile, (iv) file operations. The necessary information about splitting a program into multiple files (including header files), using make and makefile, and file operations have already been presented in class. Information about Directories and the directory hierarchy will be presented in class next week.

B. Description. The executable version of your program must be named `fs`. Your `makefile` must ensure this. The `fs` program must support the following usage:

```
fs [-r] [-d] [-b|-k] ‘‘filepattern’’
```

If no switches are given, your `fs` program should report the size (in bytes, as reported by `stat`) of all files relative to the current directory that match the file patterns, one per line. You may assume that any argument that begins with a dash (-) is a command line switch, and all switches precede any file patterns.

The command line parameter `filepatern` represents the pattern that files need to match in order for their size to be reported by the fs program. If no file patterns are given, or if no files match the given patterns, your program should produce no output. For the purposes of this assignment, a pattern will be identified by a short text string that complies with the following pattern syntax:

- You can assume that filenames are defined as any sequence of consecutive alphanumeric characters (a-z, A-Z, 0-9). There is no need to check for illegal characters in your program.
- You can assume that only non-hidden files are considered.

- For a literal match (i.e., exact match, wrap the name of the file to be matched in brackets. For example, “[foo.c]” matches the file foo.c.
- If the first character after the opening quotes character (i.e., “) is a star (*), it specifies a wildcard notation, i.e., a pattern that can start with any character sequence. In other words, it specifies a *suffix pattern*. For example, “*.c” matches all C files.
- If the last character before the closing quotes character (i.e., “) is a star (*), it specifies a pattern that can end with any character sequence. In other words, it specifies a *prefix pattern*. For example, “foo.*” matches all files named foo (e.g., “foo.c”, “foo.exe”, “foo.txt” etc.).
- If the user wants to mach a file containing a string, she can specify that by enclosing the string with star (*) characters. For example, the “*oo*” pattern matches all files named foo (e.g., “foo.c”), zoo (e.g., “zoo.txt”) etc. as well as files named “oops”, “ooip”, or “oo”.
- The “*.*” pattern matches any filename.

For cases not covered by this specification (such as error handling), you may specify and implement a reasonable behavior.

If the `-r` switch is given, your `fs` program should operate recursively. Normally, a `filepattern` should be matched against the entire file name of all files in the current repository. Instead, when the `-r` option is provided your `fs` program should report not only files relative to the current directory, but also any files relative to any subdirectories, and so on. You can assume that only non-hidden subdirectories are considered.

If the `-d` switch is given, your `fs` program should also report the size of ordinarily directories in addition to regular files.

If the `-b` switch is given, the number of blocks (as reported by `stat`) occupied by each file should be reported instead of its size. At most one of `-b` or `-k` (see below) may be specified.

If the `-k` switch is provided as an argument to your `-k` orfs program, sizes should be reported in kilobytes instead of bytes. Recall that there are $1,024$ bytes in a kilobyte. You may choose and document a suitable rounding convention.

Your program should report the file (or directory) sizes based on the following format: file size (or number of blocks), followed by a tab character, followed by the path to the file relative to the current directory. See the examples below for details.

Your program must detect the following fatal errors. In each case, your program should produce a suitable error message to `stderr` and stop.

- The number of command line arguments is less than two or more than 5.
- A `filepattern` is not specified.
- A `filepattern` does not comply to the pattern syntax rules specified above.
- An unknown switch is provided.

C. Examples.

The following command reports the size of all C files in the current directory.

```
$>fs "*.c"
162       a.c
938       b.c
```

The following command reports the size of all C files in all directories including and below the current directory.

```
$>fs -r "*.c"
162       a.c
938       b.c
2490      subdir/xyz.c
727       subdir/readn.c
661       subdir/subdir2/writen.c
```

The following command reports the size of all files starting with s in the current directory.

```
$>fs "s*"
31424     sol.txt
162       s.c
```

The following command repeats this, but including the subdir directory because of the -d flag.

```
$>fs -d "'s*"
31424      sol.txt
512        subdir
```

The following command reports the number of blocks in all files that contain the characters 'sa' in their filename.

```
$>fs -b -r "*sa*"
2    subdir/sail.c
4    subdir/salty.txt
2    subdir/subdir2/asap.txt
4    subdir3/haltsa.c
```

D. Structural Requirements. Your submission must have *at least two* C source files, zero or more header files, and a `makefile`. Additional requirements on the C source files are as follows.

- One source file must contain just the `main` function.
- A second source file must contain only the function that reports the size of a file (or directory).

The C files (i.e., files with extension "`.c`"), header files (i.e., files with extension "`.h`") and the `makefile` must be submitted together using the `turnin-csi402` command. Instructions for using the `turnin-csi402` command have been provided in Programming Assignment 0, which you should have already submitted.

E. Hints and Clarifications.

- Try to get the basic functionality working first before adding extra features. For instance, get the size of files in the current directory working before adding -b or -k functions, and then make it recursive.
- Explore the man page for `stat`.
- Generally, you should only need the commands that we have covered in class.
- Do not use `find` or `ls` in your solution; instead use recursion and the `stat` tool.
- For full marks, your makefile should produce an executable named `fs` when run with the `make` command.
- For full marks, your `fs` program must be well-documented and easily understandable.