```
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <dirent.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <unistd.h>

/*
Name:       Dylan Calvin
BlazerId:   dylcal13
Project #:  2
To compile: gcc hw2.c -o hw2
To run:     ./hw2 [S] [-s <size>] [-f <pattern>] <dirname>
*/


// Modified from Lab5 for the purposes of HW2.

struct flags {
    int print_file_size;
    int sort_by_file_size;
    int file_size;
    int sort_by_name;
    char *name;
};

void printFile(char *combinedPath, struct dirent *dirent, struct flags opts, struct stat
sb, int level){
    // printFile contains all the logic to handle the formatting (and optflags provided)
when printing out files and directories
    //  The inputs are the absolute path to the current file, the dirent of the current f
ile, the opts structure given from the main function,
    //    the stat structure of the current file, & the current indent level.
    // There is no ouput since all this function does is print.


    if((*dirent).d_type == DT_DIR){
        // NOTE: We are not accounting for (or sorting by) directory size, since all dire
ctories report a 4KB size.
        if(opts.sort_by_name == 1){
            if(strstr((*dirent).d_name,opts.name)){
                int j;
                for(j=0;j<level;j++){
                    printf("   ");
                }
                if(opts.print_file_size==1){
                    printf ("[D] %s (%d b)\n", (*dirent).d_name, sb.st_size);
                }
                else{
                    printf ("[D] %s\n", (*dirent).d_name);
                }
            }
        }
        else{
            int j;
            for(j=0;j<level;j++){
                printf("   ");
            }
            if(opts.print_file_size==1){
                printf ("[D] %s (%d b)\n", (*dirent).d_name, sb.st_size);
            }
            else{
                printf ("[D] %s\n", (*dirent).d_name);
            }
        }

    }
```

```
    else if((*dirent).d_type == DT_LNK){
        // Got readlink from the man pages:
        //   https://linux.die.net/man/2/readlink
        char *linkname = malloc(sb.st_size+1);
        if(readlink(combinedPath,linkname,sb.st_size+1)!=-1){
            linkname[sb.st_size]='\0';
            if(opts.sort_by_file_size == 1 && opts.sort_by_name == 1){
                if(sb.st_size >= opts.file_size && strstr((*dirent).d_name,opts.name)){
                    int j;
                    for(j=0;j<level;j++){
                        printf("   ");
                    }
                    if(opts.print_file_size==1){
                        printf ("[F] %s (links to %s) (%d b)\n", (*dirent).d_name, linkna
me, sb.st_size);
                    }
                    else{
                        printf ("[F] %s (links to %s)\n", (*dirent).d_name, linkname);
                    }
                }
            }
            else if(opts.sort_by_file_size == 1 && opts.sort_by_name == 0){
                if(sb.st_size >= opts.file_size){
                    int j;
                    for(j=0;j<level;j++){
                        printf("   ");
                    }
                    if(opts.print_file_size==1){
                        printf ("[F] %s (links to %s) (%d b)\n", (*dirent).d_name, linkna
me, sb.st_size);
                    }
                    else{
                        printf ("[F] %s (links to %s)\n", (*dirent).d_name, linkname);
                    }
                }
            }
            else if(opts.sort_by_file_size == 0 && opts.sort_by_name == 1){
                if(strstr((*dirent).d_name,opts.name)){
                    int j;
                    for(j=0;j<level;j++){
                        printf("   ");
                    }
                    if(opts.print_file_size==1){
                        printf ("[F] %s (links to %s) (%d b)\n", (*dirent).d_name, linkna
me, sb.st_size);
                    }
                    else{
                        printf ("[F] %s (links to %s)\n", (*dirent).d_name, linkname);
                    }
                }
            }
            else if(opts.sort_by_file_size == 0 && opts.sort_by_name == 0){
                int j;
                for(j=0;j<level;j++){
                    printf("   ");
                }
                if(opts.print_file_size==1){
                    printf ("[F] %s (links to %s) (%d b)\n", (*dirent).d_name, linkname,
sb.st_size);
                }
                else{
                    printf ("[F] %s (links to %s)\n", (*dirent).d_name, linkname);
                }
            }
        }

    }
    else{
        // found strstr function in man pages, see link:
        //  https://linux.die.net/man/3/strstr
```

```
        if(opts.sort_by_file_size == 1 && opts.sort_by_name == 1){
            if(sb.st_size >= opts.file_size && strstr((*dirent).d_name,opts.name)){
                int j;
                for(j=0;j<level;j++){
                    printf("   ");
                }
                if(opts.print_file_size==1){
                    printf ("[F] %s (%d b)\n", (*dirent).d_name, sb.st_size);
                }
                else{
                    printf ("[F] %s\n", (*dirent).d_name);
                }
            }
        }
        else if(opts.sort_by_file_size == 1 && opts.sort_by_name == 0){
            if(sb.st_size >= opts.file_size){
                int j;
                for(j=0;j<level;j++){
                    printf("   ");
                }
                if(opts.print_file_size==1){
                    printf ("[F] %s (%d b)\n", (*dirent).d_name, sb.st_size);
                }
                else{
                    printf ("[F] %s\n", (*dirent).d_name);
                }
            }
        }
        else if(opts.sort_by_file_size == 0 && opts.sort_by_name == 1){
            if(strstr((*dirent).d_name,opts.name)){
                int j;
                for(j=0;j<level;j++){
                    printf("   ");
                }
                if(opts.print_file_size==1){
                    printf ("[F] %s (%d b)\n", (*dirent).d_name, sb.st_size);
                }
                else{
                    printf ("[F] %s\n", (*dirent).d_name);
                }
            }
        }
        else if(opts.sort_by_file_size == 0 && opts.sort_by_name == 0){
            int j;
            for(j=0;j<level;j++){
                printf("   ");
            }
            if(opts.print_file_size==1){
                printf ("[F] %s (%d b)\n", (*dirent).d_name, sb.st_size);
            }
            else{
                printf ("[F] %s\n", (*dirent).d_name);
            }
        }
    }
}

void printDirContents(char* parentPath, struct flags opts, int level){
    // printDirContents handles the actual crawling of the tree, by going into any direct
ory it encounters
    struct dirent *dirent;
    DIR *parentDir;
    parentDir = opendir (parentPath);
    if (parentDir == NULL) {
        printf ("Error opening directory '%s' : %s.\n", parentPath, strerror(errno));
        exit (-1);
    }
    int count = 1;
    while((dirent = readdir(parentDir)) != NULL){
        if(strcmp((*dirent).d_name,".") && strcmp((*dirent).d_name,"..")){
```

```
            // making the absolute path to give to the recursive call & print function
            char* combinedPath = malloc(sizeof(parentPath)+1+sizeof((*dirent).d_name));
            strcpy(combinedPath,parentPath);
            strcat(combinedPath,"/");
            strcat(combinedPath,(*dirent).d_name);
            struct stat sb;
            if(stat(combinedPath,&sb) == 0){
                if((*dirent).d_type == DT_DIR){
                    printFile(combinedPath,dirent,opts,sb,level);
                    //printf("%s\n",combinedPath);
                    int temp_level = level + 1;
                    printDirContents(combinedPath,opts,temp_level);
                    free(combinedPath);
                }
                else{
                    printFile(combinedPath,dirent,opts,sb,level);
                }
                count++;
            }
        }
    }
    closedir(parentDir);
}

int main (int argc, char **argv) {
    struct dirent *dirent;
    DIR *parentDir;
    struct flags opts;

    opts.print_file_size = 0;
    opts.sort_by_name = 0;
    opts.sort_by_file_size = 0;
    opts.file_size = 0;

    char *path = argv[argc-1];

    // no path, no options given
    if (argc < 2) {
        path = malloc(2048);
        getcwd(path,2048);
        printf("%s\n",path);
    }
    // has options, no path (special case: no path given & optarg flag given at end)
    if((argc > 2 && strstr(argv[argc-2],"-"))){
        path = malloc(2048);
        getcwd(path,2048);
    }
    // has options, no path (special case: no path given & last argv is -S)
    else if(strstr(argv[argc-1],"-S")){
        path = malloc(2048);
        getcwd(path,2048);
    }

    int opt;
    while((opt = getopt(argc,argv,"Ss:f:")) != -1){
        //printf("%s\n", opt);
        switch(opt){
            case 'S':
                opts.print_file_size = 1;
                break;
            case 's':
                opts.sort_by_file_size = 1;
                opts.file_size = atoi(optarg);
                break;
            case 'f':
                opts.sort_by_name = 1;
                opts.name = optarg;
                break;
            // Anything that isn't recognized is ignored (Not including dir name)
        }
```

```
    }
    // test open the directory, if it doesnt work either:
    //    1. The file name given is incorrect, and the dir does not exist
    //    2. No file name was given
    DIR *testDir;
    testDir = opendir(path);
    if (testDir == NULL) {
        printf ("Error: Please check that the given directory exists & is typed correctly
.\n", argv[0]);
        printf ("Usage: %s [S] [-s <size>] [-f <pattern>] <dirname>\n", argv[0]);
        exit(-1);
    }
    closedir(testDir);

    printDirContents(path, opts, 0);
    return 0;
}
```