```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <getopt.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>
#include <limits.h>
#include <libgen.h>
#include <sys/wait.h>

/*
Name:Shreyas Srinivasa
BlazerId: SSRINIVA
Project #: HW03
To compile: cc search.c -o search OR make
To run: ./search [path] [-t nsecs] [-S] [-s <file size in bytes>] [-f <string pattern> <depth>]
*/

void recursiveFileTraversal(char *path, int depth);

// Filter Flags/Values
int bytefilter = 0;
int fulldetails = 0;
char stringpattern[100000];
char command[100000];
int depthfilter = 0;

int main(int argc, char *argv[])
{

    char path[100000];
    int opt;

    // Reading command line arguments, setting filter data based on options and passed option and non-option arguments
    while ((opt = getopt(argc, argv, "-:Ss:f:e:")) != -1)
    {
        switch (opt)
        {
        case 'S':
            printf("Option S was provided\n");
            fulldetails = 1;
            break;
        case 's':
            printf("Option s has arg: %s\n", optarg);
            bytefilter = atoi(optarg);
            break;
        case 'f':
            printf("Option f has arg %s\n", optarg);
            strcpy(stringpattern, optarg);
            break;
        case 'e':
            printf("Option e has arg %s\n", optarg);
            strcpy(command, optarg);
            break;
        case 1:
```

```
            case 2:
                printf("Non-option arg: %s\n", optarg);
                if (strcmp(stringpattern, "") == 0)
                {
                    strcpy(path, optarg);
                }
                else
                    depthfilter = atoi(optarg);
                break;
            case '?':
                fprintf(stderr, "Usage: %s [path] [-t nsecs] [-S] [-s <file size in
bytes>] [-f <string pattern> <depth>] \n", "./search");
                exit(EXIT_FAILURE);
            case ':':
                printf("Missing arg for %c\n", optopt);
                break;
            }
        }

        // If no path is provided
        if (strcmp(path, "") == 0)
            strcpy(path, "./");

        printf("Path: %s Full Details : %d Bytes: %d String Pattern: %s Depth Filter: %d
Command: %s\n\n", path, fulldetails, bytefilter, stringpattern, depthfilter, command);

        // Printing starting directory
        printf("%s\n", basename(path));
        recursiveFileTraversal(path, 1);

        return 0;
    }

    // Helper function to check if file is a symbolic link
    int is_symlink(const char *filename)
    {
        struct stat p_statbuf;

        if (lstat(filename, &p_statbuf) < 0)
        {
            perror("calling stat()");
            exit(1);
        }

        return S_ISLNK(p_statbuf.st_mode) == 1;
    }

    // Helper function to print all file properties
    void printFileProperties(struct stat stats)
    {
        struct tm dt;
        char lastaccesstime[100000];

        printf("(");

        // r - Read, w - Write, x -Execute
        if (stats.st_mode & R_OK)
            printf("r");
        else
            printf("-");
```

```
        if (stats.st_mode & W_OK)
            printf("w");
        else
            printf("-");
        if (stats.st_mode & X_OK)
            printf("x");
        else
            printf("-");

        // Size 0 for directories, otherwise size is displayed in bytes
        if (S_ISDIR(stats.st_mode))
            printf(", %d", 0);
        else
            printf(", %zd", stats.st_size);

        // Prints last accessed time in local format
        strftime(lastaccesstime, sizeof(lastaccesstime), "%c",
    localtime(&stats.st_atime));
        printf(", %s", lastaccesstime);

        printf(")");
    }

    // fork, exec and wait flow if -e option is provided
    void forkandexec(char *command, char *path)
    {
        char *tokens[1024];
        int i = 0;
        char commandCopy[1024];

        pid_t pid;
        int status;

        strcpy(commandCopy, command);

        tokens[i] = strtok(commandCopy, " ");

        while (tokens[i] != NULL)
            tokens[++i] = strtok(NULL, " ");

        tokens[i] = path;
        tokens[++i] = NULL;

        // printf("%s %s %s\n", tokens[0], tokens[1], tokens[2]);

        pid = fork();
        if (pid == 0)
        {
            execvp(tokens[0], tokens);
            perror("execv");
            exit(-1);
        }
        else if (pid > 0)
        {

            wait(&status);

            if (WIFEXITED(status))
            {
                // printf("Child process exited with status = %d\n", WEXITSTATUS(status));
```

```
            }
            else
            {
                printf("Child process did not terminate normally!\n");
            }
        }
        else
        {
            perror("fork");
            exit(EXIT_FAILURE);
        }
    }

    // Recursive function to search file tree
    void recursiveFileTraversal(char *basePath, int depth)
    {
        char path[100000];
        char linkedFile[100000];

        struct dirent *dp;
        struct stat stats;
        DIR *dir = opendir(basePath);

        if (!dir)
            return;

        while ((dp = readdir(dir)) != NULL)
        {
            if (strcmp(dp->d_name, ".") != 0 && strcmp(dp->d_name, "..") != 0)
            {

                int skipPrint = 0;

                // Preparing path for next function call
                strcpy(path, basePath);
                strcat(path, "/");
                strcat(path, dp->d_name);

                // Result of stat() - Indicates success or error
                int statResult = stat(path, &stats);
                //-f Logic
                if (strcmp(stringpattern, "") != 0 && (strstr(dp->d_name, stringpattern)
    == NULL || depth > depthfilter))
                    skipPrint = 1;
                //-s Logic
                if (bytefilter > 0 && stats.st_size > bytefilter &&
    S_ISREG(stats.st_mode))
                    skipPrint = 1;

                // Prints all file details
                if (skipPrint == 0)
                {
                    for (int i = 0; i < depth; i++)
                        putchar('\t');
                    printf("%s", dp->d_name);

                    // Check for symlinks
                    if (is_symlink(path) == 1 && readlink(path, linkedFile, sizeof
    linkedFile) != -1)
                        printf("(%s)", linkedFile);
                    // Check for fulldetails filter
```

```
                if (fulldetails == 1 && statResult == 0)
                    printFileProperties(stats);
                if (strcmp(command, "") != 0 && S_ISREG(stats.st_mode))
                    forkandexec(command, path);

                printf("\n");
            }

            recursiveFileTraversal(path, depth + 1);
        }
    }

    closedir(dir);
}
```