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

/*
Name: Elle Lohning
BlazerId: glohning
Project #: Homework 3
To compile: gcc hw3.c -g -o hw3
To run: ./<name of executable> <commands and arguments>
ex: ./hw3 -e "ls -l"
*/

typedef struct
{
   int S_flag;          // is the S flag provided?
   int s_flag;          // is the s flag provided?
   int f_flag;          // is the f flag provided?
   int t_flag;          // is the t flag provided?
   int e_flag;
   int fileSize;        // s flag value
   char filterTerm[300]; // f flag value
   char fileType[2];     // t flag value
   char command[300];
```

```
    char flag[5];
} FlagArgs;

// type definition of the function pointer. It's void because it won't return anything
typedef void FileHandler(char *filePath, char *dirfile, FlagArgs flagArgs, int nestingCount);

// the function that will be used for this assignment
void myPrinterFunction(char *filePath, char *dirfile, FlagArgs flagArgs, int nestingCount)
{
    struct stat buf;       // buffer for data about file
    lstat(filePath, &buf); // very important that you pass the file path, not just file name
    char line[100];        // init some memory for the line that will be printed
    strcpy(line, "");      // verify a clean start
    strcat(line, dirfile); // init the line with the file name

    if (flagArgs.S_flag) // S case
    {
        char strsize[10];                       // allocate memory for the string format of the size
        sprintf(strsize, " %d", (int)buf.st_size); // assign the size to the allocated string
        strcat(line, strsize);                  // concatenate the line and the size
    }
    if (flagArgs.s_flag) // s case
    {
        if (flagArgs.fileSize > (int)buf.st_size) // if the file size is less than the expected
        {
            strcpy(line, ""); // clear the line print
        }
    }
    if (flagArgs.f_flag) // f case
```

```
{
   if (strstr(dirfile, flagArgs.filterTerm) == NULL) // if the filter does not appear in the file
   {
      strcpy(line, ""); // clear the line print
   }
}
if (flagArgs.t_flag) // t case
{
   if (strcmp(flagArgs.fileType, "f") == 0) // if the provided t flag is "f"
   {
      if (S_ISDIR(buf.st_mode) != 0) // if the file is a dir
      {
         strcpy(line, ""); // clear the line print
      }
   }
   if (strcmp(flagArgs.fileType, "d") == 0) // if the provided t flag is "d"
   {
      if (S_ISREG(buf.st_mode) != 0) // if the file is a regular file
      {
         strcpy(line, ""); // clear the line print
      }
   }
}
if (flagArgs.e_flag) // e case
{
   int pid = fork();
   if (pid == 0)
   {
      printf("%s", filePath);
```

```
            execlp(flagArgs.command, flagArgs.command, flagArgs.flag, (char *)0);
        }
        else
        {
            strcpy(line, ""); // clear the line print
        }
    }
    if (strcmp(line, "") != 0) // check to prevent printing empty lines
    {
        int i = 0;
        for (i = 0; i <= nestingCount; i++) // tab printer
        {
            printf("\t"); // print a tab for every nesting
        }
        printf("%s\n", line); // print the line after the tabs
    }
}

void readFileHierarchy(char *dirname, int nestingCount, FileHandler *fileHandlerFunction, FlagArgs
flagArgs)
{
    struct dirent *dirent;
    DIR *parentDir = opendir(dirname); // open the dir
    if (parentDir == NULL)          // check if there's issues with opening the dir
    {
        printf("Error opening directory '%s'\n", dirname);
        exit(-1);
    }
    while ((dirent = readdir(parentDir)) != NULL)
```

```
    {
        if (strcmp((*dirent).d_name, "..") != 0 &&
            strcmp((*dirent).d_name, ".") != 0) // ignore . and ..
        {
            char pathToFile[300];                                   // init variable of the path to the current file
            sprintf(pathToFile, "%s/%s", dirname, ((*dirent).d_name));          // set above variable to be the path
            fileHandlerFunction(pathToFile, (*dirent).d_name, flagArgs, nestingCount); // function pointer call


            if ((*dirent).d_type == DT_DIR)                         // if the file is a dir
            {
                nestingCount++;                                     // increase nesting before going in
                readFileHierarchy(pathToFile, nestingCount, fileHandlerFunction, flagArgs); // reccursive call
                nestingCount--;                                     // decrease nesting once we're back
            }
        }
    }
    closedir(parentDir); // make sure to close the dir
}


int main(int argc, char **argv)
{
    // init opt :
    int opt = 0;
    // init a flag struct with 0s
    FlagArgs flagArgs = {
        .S_flag = 0,
        .s_flag = 0,
```

```
    .f_flag = 0,
    .t_flag = 0,
    .e_flag = 0};

// Parse arguments:
while ((opt = getopt(argc, argv, "Ss:f:t:e:")) != -1)
{
    switch (opt)
    {
    case 'S':
        flagArgs.S_flag = 1; // set the S_flag to a truthy value
        break;

    case 's':
        flagArgs.s_flag = 1;            // set the s_flag to a truthy value
        flagArgs.fileSize = atoi(optarg); // set fileSize to what was provided
        break;

    case 'f':
        flagArgs.f_flag = 1;              // set the f_flag to a truthy value
        strcpy(flagArgs.filterTerm, optarg); // set filterTerm to what was provided
        break;

    case 't':
        flagArgs.t_flag = 1;            // set the t_flag to a truthy value
        strcpy(flagArgs.fileType, optarg); // set fileType to what was provided
        break;
    case 'e':
        flagArgs.e_flag = 1;
```

```
            char temp[300];
            strcpy(temp, strtok(optarg, " "));
            strcpy(flagArgs.command, temp);
            strcpy(temp, strtok(NULL, " "));
            strcpy(flagArgs.flag, temp);
            break;
        }
    }

    if (opendir(argv[argc - 1]) == NULL) // check for if a dir is provided
    {
        char defaultdrive[300];
        getcwd(defaultdrive, 300);    // get the current working directory (if no directory was provided)
        printf("%s\n", defaultdrive); // prints the top-level dir
        readFileHierarchy(defaultdrive, 0, myPrinterFunction, flagArgs);
        return 0;
    }

    printf("%s\n", argv[argc - 1]); // prints the top-level dir
    readFileHierarchy(argv[argc - 1], 0, myPrinterFunction, flagArgs);
    return 0;
}
```