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

// char arr[][100];

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 E_flag;
    int fileSize;         // s flag value
    char filterTerm[300]; // for string in case of e -E// f flag value
    char fileType[2];     // t flag value
} FlagArgs;
int val;
char linux[10000];
char *flags[10000];

typedef void FileHandler(char *filePath, char *dirfile, FlagArgs flagArgs, int nestingCou
nt);

// the function that will be used for this assignment
void myPrinterFunction(char *filePath, char *dirfile, FlagArgs flagArgs, int nestingCount
)
{
    struct stat buf;
    lstat(filePath, &buf);
    char line[100];
    char string_eE[10];
    strcpy(line, "");
    strcat(line, dirfile);
    if (flagArgs.S_flag)
    {
        char strsize[10];
        sprintf(strsize, " %d", (int)buf.st_size);

        strcat(line, strsize);
    }
    if (flagArgs.s_flag)
    {
        if (flagArgs.fileSize > (int)buf.st_size)
        {
            strcpy(line, "");
        }
    }
    if (flagArgs.f_flag)
    {
        if (strstr(dirfile, flagArgs.filterTerm) == NULL)
        {
            strcpy(line, "");
        }
    }
    if (flagArgs.t_flag)
    {
        if (strcmp(flagArgs.fileType, "f") == 0)
        {
            if (S_ISDIR(buf.st_mode) != 0)
            {
                strcpy(line, "");
            }
        }
        if (strcmp(flagArgs.fileType, "d") == 0)
        {
            if (S_ISREG(buf.st_mode) != 0)
            {
                strcpy(line, "");
            }
        }
    }

    if (strcmp(line, "") != 0)
    {

        if (flagArgs.e_flag == 1 || flagArgs.E_flag == 1)
        {
            if (S_ISDIR(buf.st_mode) == 0)
            {
                strcat(linux, filePath);
                strcat(linux, " ");
            }
        }
        else
        {
            printf("%s\n", line);
        }
    }
}
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 thedir
    {
        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, k, m, a, b, c;
    char *arr[argc][100];
    char l_flag[50], temp[10];
    val = 0;
    // init a flag struct with 0
    FlagArgs flagArgs = {
        .S_flag = 0,
        .s_flag = 0,
        .f_flag = 0,
        .t_flag = 0,
        .e_flag = 0,
        .E_flag = 0};
    // Parse arguments:
    strcpy(linux, " ");

    while ((opt = getopt(argc, argv, "Ss:f:t:e:E:")) != -1)
    {
        switch (opt)
        {
        case 'S':
            flagArgs.S_flag = 1; // set the S_flag to a truthy value
            printf("%s", optarg);
            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 *array = strtok(optarg, " ");

            while (array != NULL)
            {
                flags[val++] = array;
                array = strtok(NULL, " ");
            }

            break;
        case 'E':
            flagArgs.E_flag = 1;
            char *array1 = strtok(optarg, " ");

            while (array1 != NULL)
            {
                flags[val++] = array1;
                array1 = strtok(NULL, "");
            }
            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 directo
ry was provided)
        printf("%s\n", defaultdrive); // prints the top-level dir
        readFileHierarchy(defaultdrive, 0, myPrinterFunction, flagArgs);

        if (flagArgs.e_flag == 1 || flagArgs.E_flag == 1)
        {
            if (strcmp(linux, "") != 0)
            {

                char *array = strtok(linux, " ");

                while (array != NULL)
                {
                    flags[val++] = array;
                    array = strtok(NULL, " ");
                }
                flags[val] = NULL;
                // for (int k = 0; k <= val; k++)
                // {
                //     printf("%s \n", flags[k]);
                // }
                int id = fork();
                if (id == 0)
                {
                    int status = execvp(flags[0], flags);
                    if (status == -1)
                        printf("error");
                }
                else if (id > 0)
                {
                    wait(NULL);
                }
                return 0;
            }
        }
    }
    printf("%s\n", argv[argc - 1]); // prints the top-level dir
    readFileHierarchy(argv[argc - 1], 0, myPrinterFunction, flagArgs);
    if (flagArgs.e_flag == 1 || flagArgs.E_flag == 1)
    {
        if (strcmp(linux, "") != 0)
        {

            char *array = strtok(linux, " ");

            while (array != NULL)
            {
                flags[val++] = array;
                array = strtok(NULL, " ");
            }
            flags[val] = NULL;
            // for (int k = 0; k <= val; k++)
            // {
            //     printf("%s \n", flags[k]);
            // }
            int id = fork();
            if (id == 0)
            {
                int status = execvp(flags[0], flags);
                if (status == -1)
                    printf("error");
            }
            else if (id > 0)
            {
                wait(NULL);
            }
        }
    }
    return 0;
}
```