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

/*
Name: Drew Bae
BlazerID: jae912
Project#: HW3
To compile: make build
To run: ./hw3 -e "ls -la"
*/

/* #1 : Take a argument. Print out hierarchy. */
/* #1 : Completed NO ERROR. */
/* #2 : If no argument, program should print out the hierarchy it is in. */
/* #2 : Completed NO ERROR */
/* #3 : If there are other directories then the directory is first printed then the fil
es are printed with one tab indention. */
/* #3 : Kinda? */
/* #4 : Check to see if it is a symbolic link and display the name. In parentheses the
file it leads to. (file) */
/* #4 : Can't make symbolic links on Vulcan */
/* #5 : Support command-line options */
/* #5.1 : (-S) Have file size in parenthesis */
/* #5.1 : Completed HELLS YAH */
/* #5.2 : (-s) Only output files with filesize equal to greather than the argument give
n */
/* #5.2 : Completed GOT'EM */
/* #5.3 : (-f) If the argument matches the substring (png, txt, jpg) then output the fi
les */
/* #5.3 : Completed ezpz */
/* #6 : Command lines must run with other command lines. */
/* #6 : Runs all the command lines but not as intended */
char *filetype(unsigned char type)
{
    char *str;
    switch (type)
    {
    case DT_BLK:
        str = "block device";
        break;
    case DT_CHR:
        str = "character device";
        break;
    case DT_DIR:
        str = "directory";
        break;
    case DT_FIFO:
        str = "named pipe (FIFO)";
        break;
    case DT_LNK:
        str = "symbolic link";
        break;
    case DT_REG:
        str = "regular file";
        break;
    case DT_SOCK:
        str = "UNIX domain socket";
```

```
        break;
    case DT_UNKNOWN:
        str = "unknown file type";
        break;
    default:
        str = "UNKNOWN";
    }
    return str;
}

/* FUNCTION POINTER */
int fun(int a)
{
    return a;
}

/* For Regular */
int recursive(DIR *parentDir, char *path)
{
    struct dirent *dirent;
    int count = 1;
    char *duplicatePath = (char *)malloc(2000);

    parentDir = opendir(path);
    if (parentDir == NULL)
    {
        printf("Error opening directory '%s'\n", dirent->d_name);
        exit(-1);
    }
    while ((dirent = readdir(parentDir)) != NULL)
    {
        if (dirent->d_type == DT_DIR)
        {
            if (strcmp(dirent->d_name, ".") == 0 || (strcmp(dirent->d_name, "..") == 0)
)
            {
                continue;
            }

            strcpy(duplicatePath, path);
            duplicatePath = strcat(duplicatePath, "/");
            duplicatePath = strcat(duplicatePath, dirent->d_name);
            printf("[%d] %s (%s)\n", count, dirent->d_name, filetype(dirent->d_type));
            count++;
            recursive(parentDir, duplicatePath);
        }
        if (dirent->d_type != DT_DIR)
        {
            printf("    ");
            printf("[%d] %s (%s)\n", count, dirent->d_name, filetype(dirent->d_type));
            count++;
        }
    }
    closedir(parentDir);
    return 0;
}

/* For -S */
int recursiveS(DIR *parentDir, char *path)
{
    struct stat buf;
    struct dirent *dirent;
    int count = 1;
```

```
    char *duplicatePath = (char *)malloc(2000);

    parentDir = opendir(path);
    if (parentDir == NULL)
    {
        printf("Error opening directory '%s'\n", dirent->d_name);
        exit(-1);
    }
    while ((dirent = readdir(parentDir)) != NULL)
    {
        if (dirent->d_type == DT_DIR)
        {
            if (strcmp(dirent->d_name, ".") == 0 || (strcmp(dirent->d_name, "..") == 0)
)
            {
                continue;
            }
            lstat(dirent->d_name, &buf);
            strcpy(duplicatePath, path);
            duplicatePath = strcat(duplicatePath, "/");
            duplicatePath = strcat(duplicatePath, dirent->d_name);
            printf("[%d] %s (%lld)\n", count, dirent->d_name, (long long)buf.st_size);
            count++;
            recursiveS(parentDir, duplicatePath);
        }
        if (dirent->d_type != DT_DIR)
        {
            lstat(dirent->d_name, &buf);
            printf("    ");
            printf("[%d] %s (%lld)\n", count, dirent->d_name, (long long)buf.st_size);
            count++;
        }
    }
    closedir(parentDir);
    return 0;
}

/* For -s */
int recursives(DIR *parentDir, char *path, int size)
{
    struct stat buf;
    struct dirent *dirent;
    int count = 1;
    char *duplicatePath = (char *)malloc(2000);

    parentDir = opendir(path);
    if (parentDir == NULL)
    {
        printf("Error opening directory '%s'\n", dirent->d_name);
        exit(-1);
    }
    while ((dirent = readdir(parentDir)) != NULL)
    {
        if (dirent->d_type == DT_DIR)
        {
            if (strcmp(dirent->d_name, ".") == 0 || (strcmp(dirent->d_name, "..") == 0)
)
            {
                continue;
            }
            lstat(dirent->d_name, &buf);
            strcpy(duplicatePath, path);
            duplicatePath = strcat(duplicatePath, "/");
```

```
            duplicatePath = strcat(duplicatePath, dirent->d_name);
            if (size <= (long long)buf.st_size)
            {
                printf("[%d] %s\n", count, dirent->d_name);
                count++;
            }
            recursives(parentDir, duplicatePath, size);
        }
        if (dirent->d_type != DT_DIR)
        {
            lstat(dirent->d_name, &buf);
            if (size <= (long long)buf.st_size)
            {
                printf("    ");
                printf("[%d] %s\n", count, dirent->d_name);
                count++;
            }
        }
    }
    closedir(parentDir);
    return 0;
}

/* for -f */
int recursivef(DIR *parentDir, char *path, char *sub)
{
    struct dirent *dirent;
    int count = 1;
    char *duplicatePath = (char *)malloc(2000);
    char *contain = (char *)malloc(2000);

    parentDir = opendir(path);
    if (parentDir == NULL)
    {
        printf("Error opening directory '%s'\n", dirent->d_name);
        exit(-1);
    }
    while ((dirent = readdir(parentDir)) != NULL)
    {
        if (dirent->d_type == DT_DIR)
        {
            if (strcmp(dirent->d_name, ".") == 0 || (strcmp(dirent->d_name, "..") == 0)
)
            {
                continue;
            }
            strcpy(duplicatePath, path);
            duplicatePath = strcat(duplicatePath, "/");
            duplicatePath = strcat(duplicatePath, dirent->d_name);
            contain = strstr(dirent->d_name, sub);
            if (contain)
            {
                printf("[%d] %s\n", count, dirent->d_name);
                count++;
            }
            recursivef(parentDir, duplicatePath, sub);
        }
        if (dirent->d_type != DT_DIR)
        {
            contain = strstr(dirent->d_name, sub);
            if (contain)
            {
                printf("    ");
```

```
                printf("[%d] %s\n", count, dirent->d_name);
                count++;
            }
        }
    }
    closedir(parentDir);
    return 0;
}

/* for -e */
int recursivee(DIR *parentDir, char *path, char *x)
{
    struct dirent *dirent;
    int count = 1;
    char *duplicatePath = (char *)malloc(2000);
    pid_t pid;
    int status;

    parentDir = opendir(path);
    if (parentDir == NULL)
    {
        printf("Error opening directory '%s'\n", dirent->d_name);
        exit(-1);
    }
    while ((dirent = readdir(parentDir)) != NULL)
    {
        if (dirent->d_type == DT_DIR)
        {
            if (strcmp(dirent->d_name, ".") == 0 || (strcmp(dirent->d_name, "..") == 0)
)
            {
                continue;
            }
            strcpy(duplicatePath, path);
            duplicatePath = strcat(duplicatePath, "/");
            duplicatePath = strcat(duplicatePath, dirent->d_name);
            printf("[%d] %s (%s)\n", count, dirent->d_name, filetype(dirent->d_type));
            count++;
            recursivee(parentDir, duplicatePath, x);
        }
        if (dirent->d_type != DT_DIR)
        {
            pid = fork();
            if (pid == 0)
            {
                char *arr[] = {"ls", "-la", NULL};
                execv("/bin/ls", arr);
            }
            else
            {
                wait(&status);
                printf("    ");
                printf("[%d] %s (%s)\n", count, dirent->d_name, filetype(dirent->d_type
));
                count++;
            }
        }
    }
    closedir(parentDir);
    return 0;
}

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

```
{
    DIR *parentDir;
    char *path, *sub, *x;
    int opt, size;

    while ((opt = getopt(argc, argv, "S S: s: f: e::")) != -1)
    {
        switch (opt)
        {
        case 'S':
            if (optarg == NULL)
            {
                parentDir = opendir(".");
                path = ".";
                if (parentDir == NULL)
                {
                    printf("Error opening directory '%s'\n", path);
                    exit(-1);
                }
                recursiveS(parentDir, path);
            }
            else
            {
                parentDir = opendir(optarg);
                path = optarg;
                if (parentDir == NULL)
                {
                    printf("Error opening directory '%s'\n", path);
                    exit(-1);
                }
                recursiveS(parentDir, path);
            }
            break;
        case 's':
            parentDir = opendir(".");
            path = ".";
            if (parentDir == NULL)
            {
                printf("Error opening directory '%s'\n", path);
                exit(-1);
            }
            size = atoi(optarg);
            recursives(parentDir, path, size);
            break;
        case 'f':
            parentDir = opendir(".");
            path = ".";
            if (parentDir == NULL)
            {
                printf("Error opening directory '%s'\n", path);
                exit(-1);
            }
            sub = optarg;
            recursivef(parentDir, path, sub);
            break;
        case 'e':
            parentDir = opendir(".");
            path = ".";
            if (parentDir == NULL)
            {
                printf("Error opening directory '%s\n", path);
                exit(-1);
            }
```

```
            x = optarg;
            recursivee(parentDir, path, x);
            break;
        }
    }

    /* USED FUCNTION POINTER HERE */
    int (*fun_ptr)(int) = &fun;
    if (argc < (*fun_ptr)(2))
    {
        parentDir = opendir(".");
        path = ".";
    }
    else
    {
        parentDir = opendir(argv[1]);
        path = argv[1];
    }
    if (parentDir == NULL)
    {
        printf("Error opening directory '%s'\n", argv[1]);
        exit(-1);
    }
    recursive(parentDir, path);
    return 0;
}
```