```
//  Camille Chow
//  hunt.c
//  ECE357 Assignment 2
//  10/8/17

#include <sys/stat.h>
#include <dirent.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <limits.h>

//Global Variables
ino_t target_ino;
off_t target_size;
dev_t target_dev;
char* target_name;
const int buff_size = BUFSIZ;

//Function to compare contents of files
//returns 1 if identical
//returns 0 if not identical
//returns -1 on error
int compareFiles(char* path) {

    int fd1, fd2;
    char* b1[buff_size], b2[buff_size];

    //open files
    if ((fd1 = open(target_name, O_RDONLY)) < 0) {
        fprintf(stderr, "Warning: Can't open target file '%s' for reading: %s
               \nCan't check for duplicate\n", target_name, strerror(errno));
        return -1;
    }
    if ((fd2 = open(path, O_RDONLY)) < 0) {
        fprintf(stderr, "Warning: Can't open file '%s' for reading: %s\nCan't
               check if duplicate\n", path, strerror(errno));
        return -1;
    }

    int n,m;
    //read to buffer
    while ((n = read(fd1, b1, buff_size)) != 0 && (m = read(fd2, b2, buff_size))
          != 0) {
        if (n < 0) {
            fprintf(stderr, "Error reading from target file '%s': %s\nCan't check
                    for duplicate\n", target_name, strerror(errno));
            return -1;
        }
        else if (m < 0) {
            fprintf(stderr, "Error reading from target file '%s': %s\nCan't check
                    if duplicate\n", path, strerror(errno));
            return -1;
        }
```

```
        else if (memcmp(b1,b2,n) != 0) {
            return 0;
        }
    }
    return 1;
}

//Recursive searching function
void searchFiles(char *directory, int canTraverse) {

    DIR *dir;
    struct dirent *entry;

    if (!(dir = opendir(directory))) {
        fprintf(stderr, "Warning: Could not open directory %s: %s\n", directory,
                strerror(errno));
        return;
    }

    while ((entry = readdir(dir)) != NULL) {

        //current path
        char path[PATH_MAX];
        sprintf(path, "%s/%s", directory, entry->d_name);

        //run stat on entry
        struct stat st;
        if (stat(path,&st) < 0) {
            fprintf(stderr, "Warning: Could not run stat on entry %s: %s\n", path
                    , strerror(errno));
            continue;
        }
        mode_t mode = st.st_mode;
        ino_t ino = st.st_ino;
        off_t size = st.st_size;
        dev_t dev = st.st_dev;

        //if entry is another directory, recursively search
        if ((mode & S_IFMT) == S_IFDIR) {
            if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") ==
                    0) {
                continue;
            }
            searchFiles(path, ((mode & S_IXOTH) == S_IXOTH) && canTraverse);
        }

        //symlink handling
        else if ((mode & S_IFMT) == S_IFLNK) {

            //find contents of link
            char link[PATH_MAX];
            if (readlink(path,link,sizeof(link)) < 0) {
                fprintf(stderr, "Warning: Could check contents of symlink %s: %s
                        \n", path, strerror(errno));
                continue;
            }
```

```
        //run stat on link
        struct stat st2;
        if (stat(link,&st2) < 0) {
            fprintf(stderr, "Warning: Could not run stat on contents of
                     symlink %s (%s): %s\n", path, link, strerror(errno));
            continue;
        }

        mode_t mode = st2.st_mode;
        ino_t ino = st2.st_ino;
        off_t size = st2.st_size;
        dev_t dev = st2.st_dev;

        if ((mode & S_IFMT) == S_IFREG) {
            //resolves to target
            if (ino == target_ino && dev == target_dev) {
                printf("%s\tSYMLINK RESOLVES TO TARGET\n",path);
            }
            //resolves to duplicate
            else if (size == target_size && compareFiles(link) == 1) {
                printf("%s\tSYMLINK (%s) RESOLVES TO DUPLICATE\n",path,link);
            }
        }
        else {
            printf("%s links to something not a file, skipping\n", path);
        }

    }
    //regular file handling
    else if ((mode & S_IFMT) == S_IFREG) {

        char* perm_string;

        if ((mode & S_IROTH) != 0 && canTraverse) {
            perm_string = "OK READ by OTHER";
        }
        else {
            perm_string = "NOT READABLE by OTHER";
        }

        //check for hardlink
        if (ino == target_ino && dev == target_dev) {
            printf("%s\tHARD LINK TO TARGET\t%s\n",path,perm_string);
        }

        //check for duplicate
        else if (size == target_size && compareFiles(path) == 1) {
            nlink_t links = st.st_nlink;
            printf("%s\tDUPLICATE OF TARGET (nlink=%d)\t%s\n",path,links,
                     perm_string);
        }
    }
    //other file type
    else {
        printf("%s not a directory, regular file, or symlink, skipping\n",
```

```
                    path);
            }
        }
        closedir(dir);
    }

    //main function
    int main(int argc, char**argv) {

        if (argc != 3) {
            fprintf(stderr, "Incorrect number of input arguments\n");
            return -1;
        }

        target_name = argv[1];
        char* starting_directory = argv[2];

        DIR* dir;
        struct stat st, st_dir;
        int t;

        if (stat(target_name,&st) < 0) {
            fprintf(stderr, "FATAL ERROR: Could not run stat on target file '%s': %s
                   \n", target_name, strerror(errno));
            return -1;
        }
        if (!(dir = opendir(starting_directory))) {
            fprintf(stderr, "FATAL ERROR: Could not open directory '%s': %s\n",
                   starting_directory, strerror(errno));
            return -1;
        }
        if (stat(starting_directory,&st_dir) < 0) {
            fprintf(stderr, "Warning: Could not run stat on directory '%s': %s\nCan't
                   determine traversal permissions\n", starting_directory, strerror
                   (errno));
            t = 0; //assume others can't traverse

        }
        else {
            mode_t mode = st_dir.st_mode;
            t = (mode & S_IXOTH) == S_IXOTH; //traversal permissions of starting
                   directory
        }

        target_ino = st.st_ino;
        target_size = st.st_size;
        target_dev = st.st_dev;

        searchFiles(starting_directory,t);

        return 0;
    }
```