```
/*******************************
*
* Sharang Phadke
* 10/04/2013
* ECE 357: OS
* Project 2 - find
* *******************************/

#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <errno.h>
#include <dirent.h>
#include <limits.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <pwd.h>
#include <grp.h>


int errsv;


//errorReport
//  function to report errors to stderr
void errorReport(int errnum, char description[], char pathname[])
{
    fprintf(stderr, "%s", strerror(errnum));
    fprintf(stderr, "\n");
    fprintf(stderr, "%s", description);
    fprintf(stderr, "%s", pathname);
    fprintf(stderr, "\n");
}


//listDir
//  recursively walk through directories beginning with
//  startPath, handling errors along the way and printing
//  the characteristics of each file and directory in
//  approximately find -ls format
void listDir(char *startPath, int uid, long mtime)
{
    char isCurOrParentDir;
    char path[PATH_MAX];
    char linkPath[PATH_MAX];
    char perm[10];
    char *t, *uFmt, *sizeFmt, *user, *group;
    DIR *dir;
    struct dirent *dirEnt;
    struct stat sp;
    struct passwd *up;
    struct group *gp;
    long nodesize;

    if(!(dir = opendir(startPath)))
    {
        errsv = errno;
        errorReport(errsv,"Error opening directory: ",startPath);
        return;
    }
    else
    {
```

```
        while (dirEnt = readdir(dir))
        {
            sprintf(path, "%s/%s", startPath, dirEnt->d_name);
            isCurOrParentDir = (strcmp(dirEnt->d_name,".") == 0)
                || (strcmp(dirEnt->d_name,"..") == 0);

            if(!isCurOrParentDir)
            {
                if(dirEnt->d_type == DT_DIR)
                    listDir(path,uid,mtime);

                if(lstat(path, &sp) == -1)
                {
                    errsv = errno;
                    errorReport(errsv,"Stat error for file: ",path);
                    continue;
                }

                //skip files by other uids
                if((uid != -1) && (uid != sp.st_uid))
                    continue;

                //skip files marked by mtime
                if(mtime >= 0)
                {
                    if((time(NULL) - sp.st_mtime) < mtime)
                        continue;
                }
                else
                {
                    if((time(NULL) - sp.st_mtime) > -mtime)
                        continue;
                }

                t = ctime(&(sp.st_mtime));

                //print device number or
                if((dirEnt->d_type == DT_BLK) || (dirEnt->d_type == DT_CHR))
                {
                    sizeFmt = "%x";
                    nodesize = sp.st_rdev;
                }
                else
                {
                    sizeFmt = "%ld";
                    nodesize = sp.st_size;
                }

                char userExecutable = (sp.st_mode & S_IXUSR);
                char grpExecutable = (sp.st_mode & S_IXGRP);

                //format permissions string
                perm[0] = S_ISBLK(sp.st_mode)
                    ? 'b'
                    : S_ISCHR(sp.st_mode)
                    ? 'c'
                    : S_ISDIR(sp.st_mode)
                    ? 'd'
                    : S_ISREG(sp.st_mode)
                    ? '-'
                    : S_ISLNK(sp.st_mode)
                    ? 'l'
                    : S_ISFIFO(sp.st_mode)
                    ? 'p'
                    : S_ISSOCK(sp.st_mode)
                    ? 's'
                    : 'w';
```

```
            perm[1] = (sp.st_mode & S_IWUSR) ? 'w' : '-';
            perm[2] = (sp.st_mode & S_IRUSR) ? 'r' : '-';
            perm[3] = ((sp.st_mode & S_ISUID) && userExecutable)
                ? 's'
                : ((sp.st_mode & S_ISUID) && !userExecutable)
                ? 'S'
                : userExecutable
                ? 'x'
                : '-';
            perm[4] = (sp.st_mode & S_IRGRP) ? 'r' : '-';
            perm[5] = (sp.st_mode & S_IWGRP) ? 'w' : '-';
            perm[6] = ((sp.st_mode & S_ISGID) && grpExecutable)
                ? 's'
                : ((sp.st_mode & S_ISGID) && !grpExecutable)
                ? 'S'
                : grpExecutable
                ? 'x'
                : '-';
            perm[7] = (sp.st_mode & S_IROTH) ? 'r' : '-';
            perm[8] = (sp.st_mode & S_IWOTH) ? 'w' : '-';
            perm[9] = (sp.st_mode & S_IXOTH) ? 'x' : '-';

            //retrieve user and group names
            if((up = getpwuid(sp.st_uid)) != NULL)
                user = up->pw_name;
            else
                sprintf(user,"%ld",(long)sp.st_uid);

            if((gp = getgrgid(sp.st_gid)) != NULL)
                group = gp->gr_name;
            else
                sprintf(group,"%ld",(long)sp.st_gid);

            //if the file is a symlink, read the link
            char symLink[PATH_MAX+5] = {0};
            if(S_ISLNK(sp.st_mode))
            {
                char linkPath[PATH_MAX] = {0};
                if(readlink(path,linkPath,sp.st_size+1) == -1)
                {
                    errsv = errno;
                    errorReport(errsv,"Error reading symbolic link: ",path);
                    continue;
                }
                strcpy(symLink, "-> ");
                strcat(symLink, linkPath);
                strcat(symLink,"\n");
            }
            else
                symLink[0] = '\n';


            //format string and print entry
            char fmt[40] = "%04ld/%ld \t %s \t %ld \t %s \t %s \t ";
            strcat(fmt, sizeFmt);
            strcat(fmt, " \t %s %s %s");

            printf(fmt,
                (long)sp.st_dev, (long)sp.st_ino, perm, (long)sp.st_nlink,
                user, group, nodesize, strndup(t,(int)strlen(t)-1), path, symLink);
        }
        errno = 0;  //set errno to 0 to check for readdir errors
    }
    if(errno != 0)
    {
        errsv = errno;
        errorReport(errsv,"Error reading directory: ",startPath);
```

```
            //if dir can be closed, continue recursive find, else exit
            if(closedir(dir) == -1)
            {
                errsv = errno;
                errorReport(errsv,"Error closing directory: ",startPath);
                exit(-1);
            }
            return;
        }
        if(closedir(dir) == -1)
        {
            errsv = errno;
            errorReport(errsv,"Error closing directory: ",startPath);
            exit(-1);
        }
    }
}


int main (int argc, char **argv)
{
    int c;
    char *startPath = ".";
    int uid = -1;
    long mtime = 0;

    //extract options and arguments
    opterr = 0;
    while ((c = getopt(argc, argv, "u:m:")) != -1)
    {
        switch (c)
        {
            case 'u':
                uid = isdigit(optarg[0])
                ? atoi(optarg)
                : getpwnam(optarg)->pw_uid;
                break;
            case 'm':
                mtime = atol(optarg);
                break;
            case '?':
                if (optopt == 'u' || optopt == 'm')
                    fprintf (stderr, "Option -%c requires an argument.\n", optopt);
                else if (isprint (optopt))
                    fprintf (stderr, "Unknown option `-%c'.\n", optopt);
                else
                    fprintf (stderr, "Unknown option character `\\x%x'.\n", optopt);
                exit(-1);
        }
    }

    if(argc > optind)
        startPath = argv[optind];

    listDir(startPath,uid,mtime);

    return 0;
}
```