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

void throwError(char *message, char *file)
{
    if (file)
        fprintf(stderr, "%s [%s]: Error code %i: %s\n", message, file, errno,
strerror(errno));
    else
        fprintf(stderr, "%s\n", message);
    exit(-1);
}

char *getPermissions(mode_t mode)
{
    char *p;
    p = (char *)malloc(10);
    if (p == NULL)
        throwError("Error: Failure to dynamically allocate memory.", NULL);
    p[0] = '-';
    p[1] = (mode & S_IRUSR) ? 'r' : '-';
    p[2] = (mode & S_IWUSR) ? 'w' : '-';
    p[3] = (mode & S_IXUSR) ? 'x' : '-';
    p[4] = (mode & S_IRGRP) ? 'r' : '-';
    p[5] = (mode & S_IWGRP) ? 'w' : '-';
    p[6] = (mode & S_IXGRP) ? 'x' : '-';
    p[7] = (mode & S_IROTH) ? 'r' : '-';
    p[8] = (mode & S_IWOTH) ? 'w' : '-';
    p[9] = (mode & S_IXOTH) ? 'x' : '-';
    return p;
}

void readDir(char *directory)
{
    DIR *dir;
    struct dirent *sd;
    struct stat buf;
    struct group *grp;
    struct passwd *usr;
    struct tm fileTime, nowTime;
    char path[4096], link[4096], date[80], *group, *user, *perms;
    mode_t mode;
    time_t now = time(NULL);

    dir = opendir(directory);
    if (dir == NULL)
    {
        throwError("Error: Unable to open directory", directory);
    }

    while ((sd = readdir(dir)) != NULL)
    {
        snprintf(path, sizeof(path), "%s/%s", directory, sd->d_name);
        if (lstat(path, &buf) < 0)
        {
            throwError("Error: Couldn't get stats for a path/file", path);
        };

        if ((grp = getgrgid(buf.st_gid)) == NULL)
        {
            throwError("Error: Could not get the group associated with a 
file/directroy", directory);
        };

        if ((usr = getpwuid(buf.st_uid)) == NULL)
        {
            throwError("Error: Could not get the user associated with a 
file/directroy", directory);
        };

        group = grp->gr_name;
        user = usr->pw_name;
        mode = buf.st_mode;

        perms = getPermissions(mode);

        localtime_r(&buf.st_mtime, &fileTime);
        localtime_r(&now, &nowTime);
        if (fileTime.tm_year == nowTime.tm_year)
        {
            strftime(date, sizeof(date), "%b %e %H:%M", &fileTime);
        }
        else
        {
            strftime(date, sizeof(date), "%b %e  %Y", &fileTime);
        }

        if (S_ISDIR(mode))
        {
            if ((strcmp(sd->d_name, "..") != 0) && (strcmp(sd->d_name, ".") != 0))
            {
                readDir(path);
            }
            else if (strcmp(sd->d_name, ".") == 0)
            {
                perms[0] = 'd';
                printf("%llu%9lli %s%5hu %s%15s%20lli %s %s\n", buf.st_ino, buf.st_blocks, perms, buf.st_nlink, user, group, buf.st_size, date, directory);
            }
        }
        else if (S_ISREG(mode))
        {
            perms[0] = '-';
            printf("%llu%9lli %s%5hu %s%15s%20lli %s %s\n", buf.st_ino, buf.st_blocks, perms, buf.st_nlink, user, group, buf.st_size, date, path);
        }
        else if (S_ISLNK(mode))
        {
            perms[0] = 'l';
            ssize_t len = readlink(path, link, 4095);
            if (len < 0)
            {
                throwError("Error: Could not read path of symbolic link", path);
            }
            else
            {
                link[len] = '\0';
            }
            printf("%llu%9lli %s%5hu %s%15s%20lli %s %s -> %s\n", buf.st_ino, buf.st_blocks, perms, buf.st_nlink, user, group, buf.st_size, date, path, link);
        }
        else
        {
            throwError("Error: This type of file is not yet supported", NULL);
        }

        free(perms);
    }
    if (closedir(dir) < 0)
    {
        throwError("Error: Unable to close directory", directory);
    }
}
```

```c

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

    char *directory = ".";

    if (argc == 1)
    {
        directory = ".";
    }
    else if (argc == 2)
    {
        directory = argv[1];
    }
    else
    {
        throwError("Error: Arguments Invalid. Correct format is './find [filepath]'", NULL);
    }
    readDir(directory);
    return 0;
}

```