Problem 1 -- A simplified find command

The unix `find (1)` command is a powerful tool, at the heart of which is an engine which can perform a recursive walk of the filesystem. In this problem set, you will write a vastly simplified version of the command which illustrates some of its functionality. It might be instructive to read the man page for `find` and explore how it works.

Your command will be invoked like this:

```
programname options starting_path
```

Your program will begin the walk at the given starting path name (which could be relative or absolute) and will recursively explore all nodes of the filesystem at and below that point. At each node, it will print out some basic information.

Don't cheat by using `fts` or `system("find")` or something like that. Your solution should involve things such as `readdir` and `stat`.

You need not be concerned with some "robustness" issues, such as running out of open file descriptors, running out of memory, etc. BUT, your program should gracefully detect and report any error conditions, as opposed to unceremoniously dumping core.

You'll find that directory entries come back to you unsorted. Don't worry about it. Also don't worry too much about a pre-order vs a post-order vs an unordered walk. All of those things would require you to sort each directory you encounter, which would only add to the unpleasantness of this assignment.

The options are, in fact optional (for the invoker of the program, not for you) and can be any one or more of the following (if you don't know how to parse command-line arguments in C, take a look at `getopt(3)`:

`-u user`: Only list nodes which are owned by the specified user, which can be given either by name or by uid number --look up the library function `getpwnam(3)`. Assume that if a number is provided, that is the uid, otherwise look up the user name.

`-m mtime`: If mtime is a positive integer, only list nodes which have NOT been modified in at least that many seconds, i.e. they are at least that many seconds old. If mtime is negative, only list nodes which have been modified no more than -mtime seconds ago. Note that `find` tends to express this in terms of days, not seconds.

At each node which is to be listed, your program will output one line in a format similar to the output of the `find` command with the `-ls` option (omitting the block count). An example output is:

```
$ ./rls /tmp/testing
0817/196265 drwxr-xr-x  4  hak      users         4096 Sep 15 2013 00:15 /tmp/testing
0817/196266 drwxr-xr-x  2  hak      users         4096 Sep 15 2013 00:16 /tmp/testing/d1
0817/196268 -rw-r--r--  1  hak      users            5 Sep 15 2013 00:16 /tmp/testing/d1/f1
0817/196270 crw-r--r--  1  root     root     0x0a14 Sep 15 2013 00:16 /tmp/testing/d2/device
0817/196271 lrwxr-xr-x  1  hak      users            8 Sep 15 2013 00:31 /tmp/testing/d2/l3
  -> ../d1/f1
```

Note the following items are required:

- The device (filesystem identifier) and inode number of the node. Note that the former is of type `dev_t`, which is a 64-bit quantity on most Linux systems. In the example above, it is printed in hex and 0-padded to at least 4 hex digits.
- The type of node and its permissions mask, in the format which ls uses. Read the man pages carefully and refer to lecture notes. Consider all of the cases (e.g. the set-gid bit is on but the group execute bit is off).
- The number of links to the node

•The owner (user) of the node. Print this out as a name, but if there is no integer->name translation available, print it out as an integer. Look at the man page `getpwuid(3)`.
•The group owner of the node. Similar deal as above.
•The size of the node, in bytes. For BLOCK SPECIAL or CHAR SPECIAL device nodes, the size field doesn't make any sense, so print the raw device number in hexadecimal instead. If you don't quite understand what this means right now, don't worry about it.
•The modification time of the node. Use any reasonable format for printing out the date/time of this timestamp. The example above differs slightly from `ls`. Look at `localtime(3)`.
•The path name of the node.
•If the node is a symbolic link, the target of the link.

Problem 2 -- Extra Credit

For 1 point extra credit, implement BOTH of the following two options:

`-x`: When specified, stay within the same mounted filesystem (we called this a "volume" in class) as the one in which your traversal began. Print a message to `stderr`,e.g.:

```
note: not crossing mount point at /dev/shm
```

`-l target`: When specified, only print out information about nodes which are symlinks whose targets successfully resolve to another node called `target`. The target need not be on the same mounted volume. E.g.

```
$ ./rls -l /tmp/testing/d1/f1 /tmp/testing
0817/196271 lrwxr-xr-x  1  hak      users            8 Sep 15 2011 00:31 /tmp/testing/d2/l3
  -> ../d1/f1
```

Caution: ruminate on the following question: For a given target T, is there a unique pathname P which resolves to that target?