Preparation For Lab 5 `stat`

To prepare for the upcoming lab, read up on the `stat` function (`man 2 stat`, and for more on it, `man 2 fstatat`). As a means of preparing our environment, let us create a file on our machine.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <unistd.h>

int main() {
    int my_fd, bytes_written;
    char *initial_contents = "this is the contents of myfile\n";

    if ((my_fd = open("myfile.txt", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR)) < 0) {
        fprintf(stderr, "Failed to create file: %s\n", strerror(errno));
        return EXIT_FAILURE;
    }
    bytes_written = write(my_fd, initial_contents, strlen(initial_contents));
    close(my_fd);

    if (bytes_written < 0) {
        fprintf(stderr, "Failed to write to file: %s\n", strerror(errno));
        return EXIT_FAILURE;
    }
    // remaining code snippets will be put in here
}
```

Now, suppose we created the file some time ago and don't already know everything about it. Suppose we want to know more information about it. Well, we can get that information with the `stat` function! Let's set that up, and fill a `struct stat` with information about the file.

```
struct stat statbuf;

// note that I had to reference statbuf to fill it. Why is this?
if (stat("myfile.txt", &statbuf) < 0) {
    fprintf(stderr, "Failed to get information on the file: %s\n", strerror(errno));
    return EXIT_FAILURE;
}
```

At this point, `statbuf` is populated with information about the file. For a full list of what information can be obtained therein, see the description in `man 2 stat`. Well, there are a few things I'm interested in related to this file.

What's the file's size?

```
printf("Size of file: %lu characters\n", statbuf.st_size);
```

What is the user id of the owner of the file?

```
printf("File owner's UID: %u\n", statbuf.st_uid);
```

When was the file last accessed? Modified? Look at `man 3 ctime` for how it is printed.

```
#include <time.h>
```

```
printf("Last access time: %s", ctime(&statbuf.st_atime));
printf("Last modification time: %s", ctime(&statbuf.st_mtime));
```

Lastly, how can I find the permissions of a file?

```
printf("Permissions of file: %d\n", statbuf.st_mode);
```

What?? That's weird, it's just some number that doesn't make a lot of sense. How can I really read into this?

Well, if you will recall, file permissions are broken up as follows:

```
| S_IRUSR    | S_IWUSR     | S_IXUSR       |
| Read owner | Write owner | Execute owner |
|            |             |               |
| S_IRGRP    | S_IWGRP     | S_IXGRP       |
| Read group | Write group | Execute group |
|            |             |               |
| S_IROTH    | S_IWOTH     | S_IXOTH       |
| Read other | Write other | Execute other |
```

These are just declarations whose values are 1 shifted some number of bits to the left. As such, we can use them to figure out the values of different permission bits in the statbuf's `mode` field, or its permission field, by performing a bitwise AND between the two. For example, after doing the following:

```
int user_read_permissions = statbuf.st_mode & S_IRUSR;
```

`user_read_permissions` will be greater than 0 if the owner can read from the file, and zero if the owner cannot read from the file. Like this, we can systematically check each bit in the permission string. Let us set up a quick example of how this might work:

```
int perms[] = {S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP, S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH};
int permission_valid;
printf("-"); // this is to print the file type bit, as displayed in `$ ls -l`.
for (int i = 0; i < 9; i += 3) {
    permission_valid = statbuf.st_mode & perms[i];
    if (permission_valid) {
        printf("r");
    } else {
        printf("-");
    }
    permission_valid = statbuf.st_mode & perms[i+1];
    if (permission_valid) {
        printf("w");
    } else {
        printf("-");
    }
    permission_valid = statbuf.st_mode & perms[i+2];
    if (permission_valid) {
        printf("x");
    } else {
        printf("-");
    }
}
printf("\n");
```

You will notice that this printout looks more familiar and human readable!