

Assignment 1: Adventure

Due Jul 7 by 11:59pm **Points** 150 **Submitting** a file upload **File Types** zip
Available Jun 22 at 8am - Jul 9 at 11:59pm 18 days

This assignment was locked Jul 9 at 11:59pm.

Introduction



In this assignment, you'll write two programs that will introduce you to programming in C on UNIX based systems and will get you familiar with reading and writing files.

- Define programming language constructs in C (Module 2, MLO 1)
- How to write a C program to solve a problem? (Module 2, MLO 2)
- How do you interact with the user in C programs? (Module 2, MLO 3)
- How is C program transformed into an executable form? (Module 2, MLO 6)
- Describe additional data types of C programming language (Module 3, MLO 1)
- Describe memory allocation in Unix (Module 3, MLO 2)
- Explain how to create and manipulate strings in C (Module 3, MLO 3)
- Describe files and directories (Module 3, MLO 4)
- Identify standard input, standard output and standard error (Module 3, MLO 5)
- Describe the API for different operations related to files (Module 3, MLO 6)
- Identify debugging tools for C programs running on Unix and use these to solve issues with your code (Module 3, MLO 7)

Instructions

Overview

This assignment is split up into two C programs (no other languages is allowed). The first program (hereafter called the "rooms program") will be contained in a file named <STUDENT ONID USERNAME>.buildrooms.c, which when compiled with the same name (minus the extension) and run creates a series of files that hold descriptions of the in-game rooms and how the rooms are connected.

The second program (hereafter called the "game") will be called <STUDENT ONID USERNAME>.adventure.c and when compiled with the same name (minus the extension) and run provides an interface for playing the game using the most recently generated rooms.

In the game, the player will begin in the "starting room" and will win the game automatically upon entering the "ending room", which causes the game to exit, displaying the path taken by the player.

Specifications

Rooms Program

The first thing your rooms program must do is create a directory called <YOUR STUDENT ONID USERNAME>.rooms.<random_number> where random_number is a random number between 0 and 100,000. Next, it must generate 7 different room files, which will contain one room per file, in the directory just created. You may use any filenames you want for these 7 room files, and these names should be hardcoded into your program. For example, if John Smith was writing the program, he might see this directory and filenames.

```
$ ls smithj.rooms.19903
Crowther_room Dungeon_room PLUGH_room PLOVER_room twisty_room XYZZY_room Zork_room
```

The elements that make up an actual room defined inside a room file are listed below, along with some additional specifications:

- A Room Name
 - A room name cannot be assigned to more than one room.
 - Each name can be at max 8 characters long, with only uppercase and lowercase letters allowed (thus, no numbers, special characters, or spaces). This restriction is not extended to the room file's filename.
 - You must hard code a list of ten different Room Names into your rooms program and have your rooms program randomly assign one of these to each room generated. Thus, for a given run of your rooms program, 7 of the 10 hard-coded room names will be used.
- A Room Type
 - The possible room type entries are: START_ROOM, END_ROOM, and MID_ROOM.
 - The assignment of which room gets which type should be randomly generated each time the rooms program is run.
 - Naturally, only one room should be assigned the START_ROOM type, and only one room should be assigned the END_ROOM type. The rest of the rooms will receive the MID_ROOM type.
- Outbound connections to other rooms
 - There must be at least 3 outbound connections and at most 6 outbound connections from this room to other rooms.
 - The outbound connections from one room to other rooms should be assigned randomly each time the rooms program is run.
 - Outbound connections must have matching connections coming back: if room A connects to room B, then room B must have a connection back to room A. Because of all of these specs, there will always be at least one path through.
 - A room cannot have an outbound connection that points to itself.
 - A room cannot have more than one outbound connection to the same room.

Each file that stores a room must have exactly this format, where the ... is additional outbound room connections, as randomly generated:

```
ROOM NAME: <room name>
CONNECTION 1: <room name>
...
ROOM TYPE: <room type>
```

Here are the contents of files representing three sample rooms from a full set of room files (remember, you must use this same format). My list of room names includes the following, among others: XYZZY, PLUGH, PLOVER, twisty, Zork, Crowther, and Dungeon.

```
ROOM NAME: XYZZY
CONNECTION 1: PLOVER
CONNECTION 2: Dungeon
CONNECTION 3: twisty
ROOM TYPE: START_ROOM


ROOM NAME: twisty
CONNECTION 1: PLOVER
CONNECTION 2: XYZZY
CONNECTION 3: Dungeon
CONNECTION 4: PLUGH
ROOM TYPE: MID_ROOM

... (Other rooms) ...

ROOM NAME: Dungeon
CONNECTION 1: twisty
CONNECTION 2: PLOVER
CONNECTION 3: XYZZY
CONNECTION 4: PLUGH
CONNECTION 5: Crowther
CONNECTION 6: Zork
ROOM TYPE: END_ROOM
```

The ordering of the connections from a room to the other rooms, in the file, does not matter. Note that the randomization you do here to define the layout is not all that important: just make sure the connections between rooms, the room names themselves and which room is which type, is somewhat different each time the rooms program is run, however you want to do that. We're not evaluating your randomization procedure, though it's not acceptable to just randomize the room names yet use the same room structure every time.

I highly recommend building up the room graph in this manner: adding outbound connections two at a time (forwards and backwards), to randomly chosen room endpoints, until the map of all the rooms satisfies the requirements listed above. It's easy, requires no backtracking, and tends to generate sparser layouts. As a warning, the alternate method of choosing the number of connections beforehand that each room will have is not recommended, as it's hard to make those chosen numbers

match the constraints of the map. To help do this correctly, please read the article [Pseudo-Code Algorithm for Generating the Random Connected Room Graph](#)  in the Resource section of the assignment and consider using the room-generating pseudo-code listed there!

Here is an example of the rooms program being compiled and then run. Note that it returns NO OUTPUT, unless there is an error:

```
$ gcc -o smithj.buildrooms smithj.buildrooms.c
$ smithj.buildrooms
$
```

The Game

Now let's describe what should be presented to the player in the game. Once the rooms program has been run, which generates the room files, the game program can then be run. This program should present an interface to the player. Note that the room data must be read back into the program from the previously generated room files, for use by the game. Since the rooms program may have been run multiple times before executing the game, your game should use the most recently created files: perform a [stat\(\) function call](#) [.\(https://linux.die.net/man/2/stat\)](https://linux.die.net/man/2/stat) on rooms directories in the same directory as the game, and open the one with most recent `st_mtime` component of the returned `stat` struct.

This player interface should list where the player current is, and list the possible connections that can be followed. Here is the form that must be used:

```
CURRENT LOCATION: XYZZY
POSSIBLE CONNECTIONS: PLOVER, Dungeon, twisty.
WHERE TO? >
```

The cursor should be placed just after the > sign. Note the punctuation used: colons on the first two lines, commas on the second line, and the period on the second line. All are required.

When the user types in the exact name of a connection to another room (Dungeon, for example), and then hits return, your program should write a new line, and then continue running as before. For example, if I typed twisty above, here is what the output should look like:

```
CURRENT LOCATION: XYZZY
POSSIBLE CONNECTIONS: PLOVER, Dungeon, twisty.
WHERE TO? >twisty

CURRENT LOCATION: twisty
POSSIBLE CONNECTIONS: PLOVER, XYZZY, Dungeon, PLUGH.
WHERE TO? >
```

If the user types anything but a valid room name from this location (case matters!), the game should return an error line that says “HUH? I DON’T UNDERSTAND THAT ROOM. TRY AGAIN.”, and repeat the current location and prompt, as follows:

```
CURRENT LOCATION: XYZZY
POSSIBLE CONNECTIONS: PLOVER, Dungeon, twisty.
WHERE TO? >Twisty

HUH? I DON'T UNDERSTAND THAT ROOM. TRY AGAIN.

CURRENT LOCATION: XYZZY
POSSIBLE CONNECTIONS: PLOVER, Dungeon, twisty.
WHERE TO? >
```

Trying to go to an incorrect location does not increment the path history or the step count. Once the user has reached the End Room, the game should indicate that it has been reached. It should also print out the path the user has taken to get there (this path should not include the start room), the number of steps taken (not the number of rooms visited, which would be one higher because of the start room), a congratulatory message, and then exit.

Note the punctuation used in the complete example below: we're looking for the same punctuation in your program.

When your program exits, set the exit status code to 0, and leave the rooms directory in place, so that it can be examined.

If you need to use temporary files (you probably won't), place them in the directory you create, above. Do not leave any behind once your program is finished. We will not test for early termination of your program, so you don't need to watch for those signals.

Complete Example

Here is a complete example of the game being compiled and run, including the building of the rooms. Note the incorrect spelling of a room name, the winning messages and formatting, and the return to the prompt with some examination commands following:

```
$ gcc -o smithj.buildrooms smithj.buildrooms.c
$ smithj.buildrooms
$ gcc -o smithj.adventure smithj.adventure.c
$ smithj.adventure
CURRENT LOCATION: XYZZY
POSSIBLE CONNECTIONS: PLOVER, Dungeon, twisty.
WHERE TO? >Twisty

HUH? I DON'T UNDERSTAND THAT ROOM. TRY AGAIN.

CURRENT LOCATION: XYZZY
POSSIBLE CONNECTIONS: PLOVER, Dungeon, twisty.
```

```
WHERE TO? >twisty

CURRENT LOCATION: twisty
POSSIBLE CONNECTIONS: PLOVER, XYZZY, Dungeon, PLUGH.
WHERE TO? >Dungeon

YOU HAVE FOUND THE END ROOM. CONGRATULATIONS!
YOU TOOK 2 STEPS. YOUR PATH TO VICTORY WAS:
twisty
Dungeon
$ echo $?
0
$ ls
smithj.adventure smithj.adventure.c smithj.buildrooms
smithj.buildrooms.c smithj.rooms.19903
$ ls smithj.rooms.19903
Crowther_room Dungeon_room PLUGH_room PLOVER_room
twisty_room XYZZY_room Zork_room
$ cat smithj.rooms.19903/XYZZY_room
ROOM NAME: XYZZY
CONNECTION 1: PLOVER
CONNECTION 2: Dungeon
CONNECTION 3: twisty
ROOM TYPE: START_ROOM
```

Hints

- You'll need to figure out how to get C to read input from the keyboard, and pause until input is received. I recommend you use the `getline()` function as described in the lectures.
- You'll also get the chance to become proficient reading and writing files. You may use either the older `open()`, `close()`, `lseek()` method of manipulating files, or the STDIO standard input library methods that use `fopen()`, `fclose()`, and `fseek()`.
- Remember that you cannot copy a string with the assignment operator (=) in C! You'll need to copy strings around using a member of the `strcpy()` family. Don't confuse string pointers with the character data itself.

Resources

For writing the code to generate the room, read the following article and consider the room-generating pseudo-code listed in it.

- Pseudo-Code Algorithm for Generating the Random Connected Room Graph to the contents of the file [CS 344 Module 2 Room Graph Generation.docx](#) 



What to Turn In

You must submit is a .zip file that contains both of your correctly named programs, which compile according to the instructions given above. If you need to make a meta-comment about this

assignment, please include it in a README file in your .zip file

Grading Criteria

The assignment is worth 15% of your grade and there are 150 points available for it.

The TAs will use this exact set of instructions: Program1 Grading Document ([PDF](#)  [WORD](#) ) to grade your submission.

Grading Comments

We expect to see comments describing what and why is happening frequently, as in every 4 or 5 lines, as appropriate. Fully commented code is worth 15 points. If there are no comments, the comment grade is 0 points. If there are comments, but not enough, give somewhere between 0 and 15 points, at your discretion.

For grading details please see the attached rubric.

Assignment 1 Rubric

Criteria	Ratings		Pts
The room generation program creates 7 room files with the correct number of room connections	28.0 pts Full Marks	0.0 pts No Marks	28.0 pts
On startup, the game program displays the correct current location	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
On startup, the game program displays the correct connection list	12.0 pts Full Marks	0.0 pts No Marks	12.0 pts
On startup, the game program displays the cursor in the correct position	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
The game program goes to the room the user types if the room is correct	15.0 pts Full Marks	0.0 pts No Marks	15.0 pts
The game program correctly identifies the case where the room entered by the user is unknown	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
When game completes, the program exits with status 0	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
When game completes, the correct number of steps are displayed	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
When game complete, the correct list of steps is displayed	15.0 pts Full Marks	0.0 pts No Marks	15.0 pts
After rooms are recreated, a different winning path can be followed	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
After rooms are recreated, a directory is created with a different name	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts

Criteria	Ratings		Pts
After rooms are recreated yet again, a different winning path can be followed	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
The code is fully commented	15.0 to >0.0 pts Full Marks	0.0 pts No Marks	15.0 pts
Total Points: 150.0			