



Introduction to Programming

High Distinction Task 1.1: Maze Game

Overview

Understanding pointers can be challenging without a good example program to help you explore the topic. In this task you will create a maze for the player to explore in a text based adventure style.

Purpose: Demonstrate that you can work with pointers to create a maze game.

Task: Complete the implementation of the Maze Game.

Time: This task should be completed before the start of week 11.

Resources:

- Programming Arcana
- Libraries and language details in cplusplus.com or other online sources

Submission Details

You must submit the following files to Doubtfire:

- Source code from the game.
- Screenshots of the game in action.

Hint: If you have problems with program crashing – make sure you are using the `my_string.str` field and not using `my_string` directly when printing or copying strings.

Instructions

In this task you will complete a maze game program that makes use of pointers and dynamic memory management. The programmer who started this was not confident with pointers, so they have started the project but left the more complex parts for you to complete.

The following steps will guide you to complete this task.

1. Download the **Maze Game** starter from Doubtfire.
2. Review the **Maze.txt** file and draw up a graph (boxes and arrows) depicting how the rooms are connected together with the various paths when this file is loaded.
3. Write up a **data dictionary** to outline the types in the supplied code.

Note: A data dictionary lists each of the custom types in a program. For enumerations it lists the different values within the types, for records it lists all of the fields and describes the what data each field will be storing.

4. Draw a **structure chart** to show the functions and procedures in the code and how they call each other.

Tip: While creating the structure chart read through the code and work out what each function and procedure is doing.

The developer has left some comments where things need to be fixed, along with some notes on what they think needs to be done.

5. Open a terminal and compile and run the program. It should run but there will be no exits in the first room.
6. Locate `// TODO: 1.`
7. At this location you need to call the `add_path` procedure. Most of the parameters are setup for you, but you need to get pointers to the *from room* and the *to room*.

Hint: To get the address of the room you need to use the `&` operator. It gives you the address of the variable to its right. For example `&my_variable`

8. Once you have this call setup correctly, switch back to the Terminal and compile and run your program. It should end with an error as `add_path` also needs fixing.
9. Locate `// TODO: 2.`
10. At this location you need to call [realloc](#). This is like [malloc](#), but allows you to re-allocate memory for your program. With `add_path` you need to re-allocate the space for the room's exits.

Hint: With `realloc` you need to pass in the original pointer and the new size eg:

```
new_values = (int*) realloc ( expenses.values ,  
                             10 * sizeof(int) );
```

This reallocates the `expenses.values` pointer to point to space for 10 integers. Notice it does not store it in `expenses.values`, as if it does not work it returns `NULL`, leaving the old memory allocation as it was.

11. Adjust the code to call `realloc`.

Note: Notice the pattern with `realloc`. The standard way of using this is to re-allocate the space to a new pointer, check that it worked, and then copy the new pointer over the old. This way if things go wrong you still have the old pointer.

12. Switch to the terminal and compile and run your program. Once it is working successfully you should see a list of exits from the first room. However, you will not be able to move to a new location.

13. Locate `// TODO: 3`.

This is the last piece of missing code. The developer was not sure how to move the player to a new room. The `player_room` is a pointer that points to the room where the player is currently located (think "which room? that room over there...").

14. Add the missing assignment statement that will change the `player_room` to point to the destination of the `player_room`'s exit that the player has chosen.

Tip: Not sure how this works? Do some hand execution.

- Draw some boxes on a piece of paper to represent the rooms.
 - Add the title (maybe skip description, its not really involved), and the array of exits.
 - In each `path_data` in `exits` add two boxes for direction and destination (we can ignore the others for now).
- Draw arrows from the destination box of a exit to a destination room.
- Draw a `player_room` variable and draw an arrow from it to the room you want to start in.
- Test your code from here...

15. Switch to the Terminal and compile and run your program. Check that you can move around the maze.

16. Relate the picture you draw in step 2 with the code structures in memory when the maze program runs.