Project: Dungeon Crawl

CMSC 216.001

Due Date: Sept 26, 2021, 11:59pm

1 Overview

Each student is asked to implement a "simple" dungeon simulation in C.

1.1 Objectives

- Process data from Standard Input (stdin)
- Allocate and manage memory dynamically
- Build and traverse arbitrarily complex data-structures in memory.

2 Detailed Description

2.1 Introduction

During the early days of the computer revolution many games featured a player exploring an underground dungeon. Examples of such games included: Rogue, Colossal Cave Adventure, and NetHack.

2.2 What to do

Create C program called "dungeon" that implements a simple Dungeon Crawler "game". Your program should take in three sets of input: a map/item specification, a starting location for the player and the dragon, and then the player's sequence of moves.

2.3 Quick Details

- infinite rooms
- infinite connections between rooms

- max 1 item/room
- player may have max 1 item
- player dies if in the room with the dragon without a sword
- dragon dies if in the room with the player with a sword
- maps need not be connected nor complete
- some connections are one-way
- you **must** use dynamic memory allocation for your main data structures.

Since there is no limit to the number of rooms or connections between them, students will have to allocate memory for rooms and connections dynamically. The selection of the appropriate dynamic memory data structures is left as an exercise for the student.

2.4 Gameplay

During the game the player will explore a potentially infinite number of rooms connected by a potentially infinite number of hallways searching for the dragon. While exploring the map your program should give the player feedback about whether he is getting closer to the dragon "warmer" or further from what does "colder". Your program should also provide varying levels of descriptive detail regarding the adjoining rooms (I suggest a list of adjoining rooms) and some indication of the distance to the dragon. The description of the relative distance to the dragon need not be a simple integer indicator, but might consist of humorous text indicative of the player's distance to the dragon. In each room the player may either choose to move to an adjacent room (by entering its room id), or pick up the item in the current room (by entering zero). If the player picks up an item while holding an existing item, the existing item will be dropped and replaced by the new item. Attempting to pick-up an item in an empty room will drop any item your are holding.

If at any time the player is in the same room as the dragon without also being in possession of a sword, the player will be eaten, lose the game, and your program should exit. If during the course of play the user should find himself in the room with the dragon and is currently in possession of the sword, he uses it to kill the dragon, the player will win the game and your program should exit.

2.5 Input Specficiation

2.5.1 Map

Consists of a potentially infinite number of lines each including a room identifier (given as an long long integer) and then a comma delimited list of adjoining rooms (given as a set of comma-separated, long long integers enclosed in parentheses), followed by a potential item in the room. Items will be string values with a length up 255 characters, and may not include the newline character. To end the input of rooms, the user should enter (invalid) room zero. You program should ignore any connections or items for room zero.

For example consider the following five room sample input:

```
1 (2,3,4) lantern
2 (3,5) rock
3 (4) axe
```

```
4 (2,3,4)
5 (5,2) sword
0 (0) ignored
```

Note: that some connections between rooms are one way (For instance room 1 is connected to room three room 3 is not connected to room 1), and it is possible for a room to be connected to itself (Rooms 4 and 5). Input order of rooms is not guaranteed to be sequential, your program should accept input in any order.

2.5.2 Starting Locations

Consists of a comma separated pair of long long integers indicating the starting location of the player and the dragon in that order. E.g.:

```
5, 1
```

In this case the player starts in room 5 and the dragon is located in room one. Your program should verify that both rooms exist in the map before continuing execution.

2.5.3 Player Moves

Are indicated as a series of long long integers separated by newlines (\n). When the player would like to pick up an item (or switch items), he should enter room zero. For example:

```
5
5
0
2
3
0
4
1
```

Note: That using the map indicated above, the player dies upon entering room one since he is in the room with the dragon, but is currently armed with an axe.

2.6 Sample Executions

2.6.1 Example 1

```
> dungeon
1 (2)
2 (1,3,5) sword
3 (4)
4 (2,3,4)
5 (5,2)
6 (4,5) lantern
0 (0)
6,1

Welcome to the dungeon.
```

```
Seriously though, someone should clean this place up.
You are in room 6, on the ground is a lantern.  Nearby are rooms 4,5.
> 0
You pick up the lantern.
You are in room 6.  Nearby are rooms 4,5.
> 4
You're getting warmer!
A metallic smell is in the air!
You are in room 4. Nearby are rooms 2,3,4.
> 0
You drop the lantern.
You are in room 4, on the ground is a lantern. Nearby are rooms 2,3,4.
> 2
You are in room 2, on the ground is a sword.  Nearby are rooms 1,3,5.
You're getting warmer!
It is stiflingly hot in this room!
> 0
You pick up the sword.
You are in room 2.  Nearby are rooms 1,3,5.
> 1
You enter a cavernous room containing piles of gold and other riches.
A large red dragon swoops down upon you from above.
You instinctively slash out with the sword, and mortally wound the beast!
You win!
```

2.6.2 Example 2

```
> dungeon
1 (1,2) rock
2 (1,3) sword
3 (2,4)
4 (3,5)
5 (4,6) machine gun
6 (5,6)
0 (0)
1,6

Welcome to the dungeon.
This seems pretty nice as far as dungeons go.
You are in room 1, on the ground is a rock. Nearby are rooms 1,2.
> 1
You're neither warmer nor colder.
This seems pretty nice as far as dungeons go.
You are in room 1, on the ground is a rock. Nearby are rooms 1,2.
> 0
You pick up the rock.
You are in room 1. Nearby are rooms 1,2.
> 2
You're getting warmer!
I've been in seedier bars then this.  I didn't stay, but it was definitely seedier.
You are in room 2, on the ground is a sword. Nearby are rooms 1,3.
You drop the rock.
You pick up the sword.
You are in room 2, on the ground is a rock. Nearby are rooms 1,3.
> 3
You're getting warmer!
Seriously though, someone should clean this place up.
You are in room 3, nearby are rooms 2,4.
> 4
You're getting warmer!
A metallic smell is in the air!
You are in room 4, nearby are rooms 3,5.
> 3
You're getting colder!
Seriously though, someone should clean this place up.
```

```
You are in room 3, nearby are rooms 2,4.
> 4
A metallic smell is in the air!
You are in room 4, nearby are rooms 3,5.
> 5
You're getting warmer!
It is stiflingly hot in this room!
You are in room 5, on the ground is a machine gun. Nearby are rooms 4,6.
> 0
You drop the sword.
You pick up the machine gun.
You are in room 5, on the ground is a sword. Nearby are rooms 4,6.
> 6
You enter a cavernous room containing piles of gold and other riches.
A large red dragon swoops down upon you from above.
The machine gun is useless against the mighty dragon.
You are burned to a crisp by its flaming breath.
(On the bright side, I hear burning alive is only like the second or third worst way to die).
You are dead.
```

2.7 Compiling you program

Please use gcc to compile and submit your program. specifically use the following command to compile your program:

```
gcc -Wall -pedantic-errors -Werror -c dungeon.c
```

All your C programs in this course should be written in correct C, which means they must compile and run correctly when compiled with the compiler `gcc`, with the options `-Wall, -pedantic-errors, and -Werror`. Except as noted below, you may use any C language features in your project that have been covered in class, or that are in the chapters covered so far and during the time this project is assigned, so long as your program works successfully using the compiler options mentioned above.

3 Submission

Your source code file must have a comment near the top that contains your name, StudentID, and M-number.

Project should be submitted via Blackboard by **September 26, 2021, 11:59pm**. Follow these instructions to turn in your project.

You should submit the following files:

- dungeon.c
- *any other source files your project needs*

The following submission directions use the command-line submit program that we will use for all projects this semester.

1. create a directory for your project:

   ```
   mkdir dungeon
   ```

2. create (or copy) all of your source files in this directory. Example: To copy a file called `example.c` into your `dungeon` directory:

```
cp example.c dungeon
```

3. change parent directory of your project directory:

```
cd dungeon/..
```

4. Create a tar file named `<user_name>.tar.gz`, where `<user_name>` is your studentID and `<proj_dir>` is the directory containing the code for your project, by typing:

```
tar -czf <user_name>.tar dungeon
```

5. Finally, submit the compressed tar file to Blackboard in accordance with the class policies.

Late assignments will not be given credit.