Lee Zong Yang 100073484

Table of Contents

1. Task 1
2. Task 2
3. Task 3
4. Task 4

**Task 1**

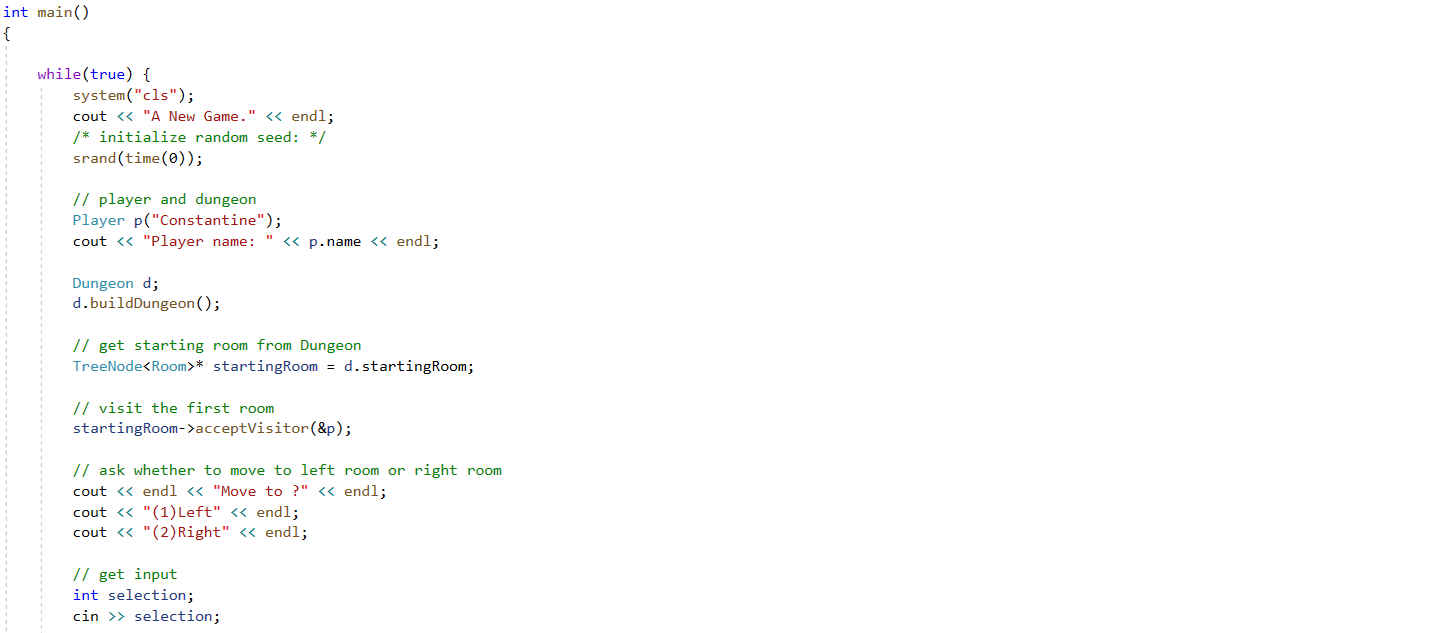
**Description**

Develop a game environment with an event loop. Define a class of player’s character and instantiate an object to represent the player during the event loop. Design room and monster in the room too. Testing should be done too.

**Concept**

I choose while loop instead of GOTO because it is a better programming practice. The reason to use a class instead of structure is due to class allow binding of data and function. Class in C++ is the building block, it allowed the software engineer to define a new custom data type. In other word, it is a blueprint for an object.

**Implementation**



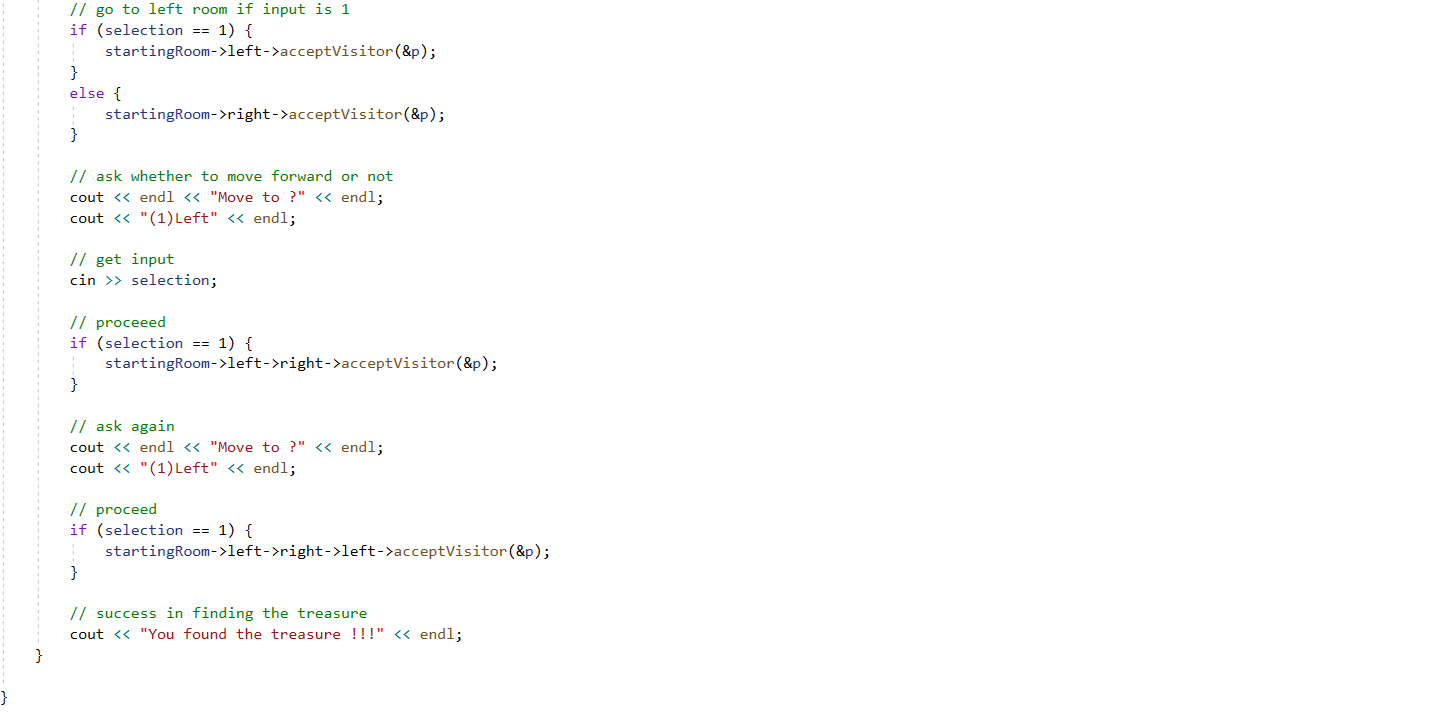


Figure: Game Loop



Figure: Monster class

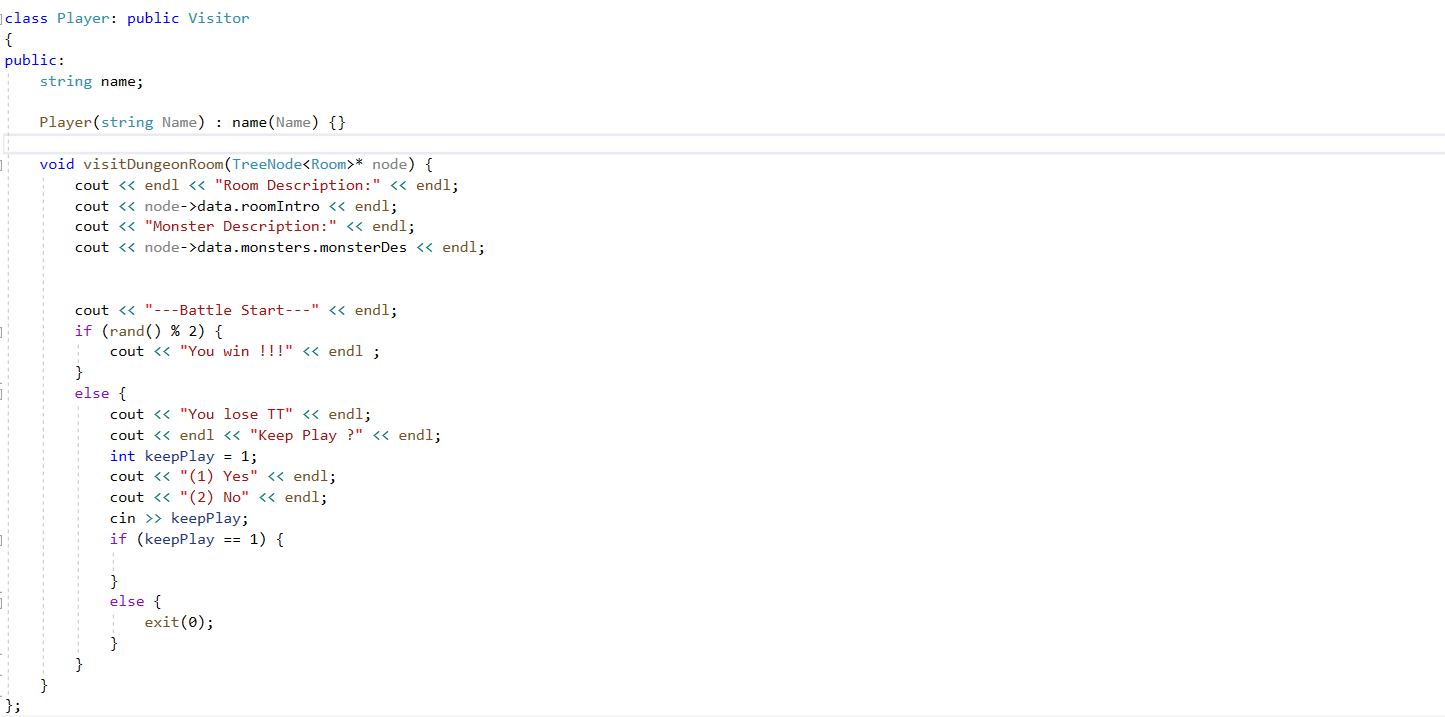


Figure: Player class

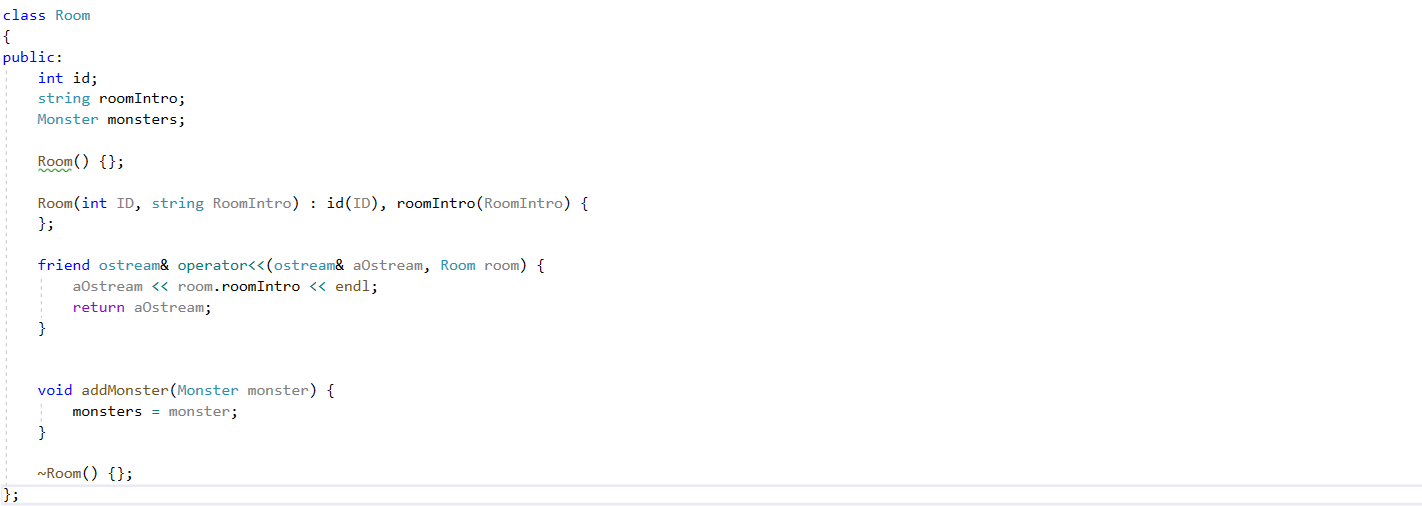


Figure: Room class

**Output**



Figure: console output

**Troubleshooting**

The design of player class is a bit difficult to accommodate visitor pattern. Otherwise, like the task I attempted before so no problem.

**Task 2**

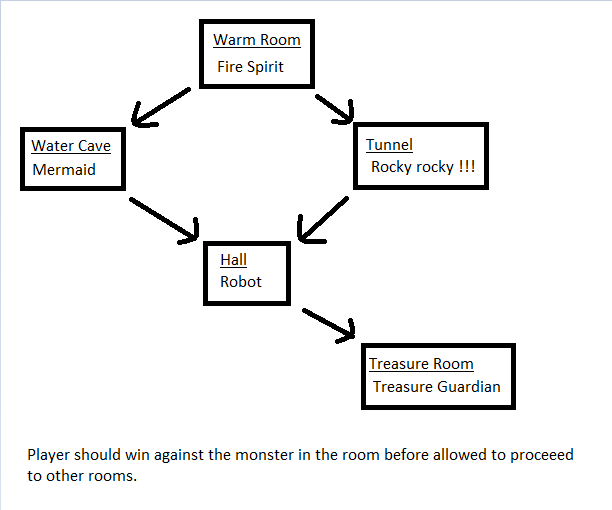
**Description**

A diagram should be produced to illustrate the dungeon. Indication of what monster each room contains and what actions are possible for the player to carry is required. As the dungeon has a tree structure, connection between rooms should be shown clearly.

**Concept**

Why uses a diagram? The main reason is the tree is a complicated concept and it is hard to visualize with text description. Humans learn and remember visually, thus the brain is designed to receive and interpret information visually. A pattern is far easier to spot than a list of numbers. Moreover, a diagram could reduce the lengthy time needed for reading and still managed to convey more information is less time.

**Implementation**



**Troubleshooting**

To find out what software is the easiest to draw a simple diagram, it took me some time to try out several software on the market. In the end, I decided to use paint as it provide all the necessary feature with simple user interface.

**Task 3**

**Description**

This task is to build the in-game dungeon. First, design a node class (node of tree) to describe the room and keep track of its entities. Second, a dungeon class to manage all these nodes. At last, testing of node traversal to show that the tree is functioning correctly.

**Concept**

A tree data structure is defined as a collection of nodes, where each node composes of its value and references to its child nodes. It is one of the important concepts in computer science, notably in Artificial Intelligence field. The nodes are connected by edges. There are different varieties of tree, binary tree is one of the famous one. It has the benefits of both an ordered array and a linked list. The search is as fast as a sorted array since binary search could be carry out. The insertion or deletion are as fast as linked list.

Root is the node at the top of the tree. Parent is a node that is one edge upward to a node. Child is a node below a given node connected by an edge downward.

**Implementation**

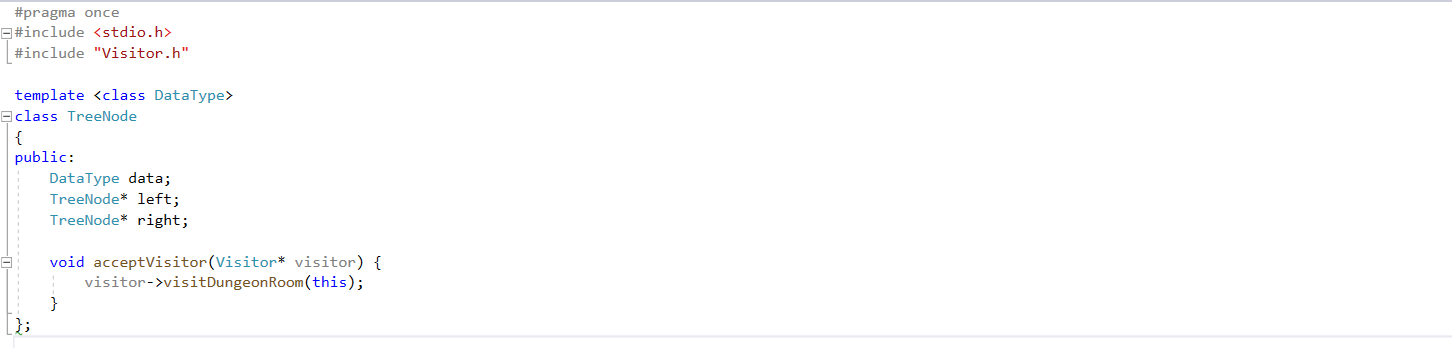


Figure: node class (TreeNode.h)

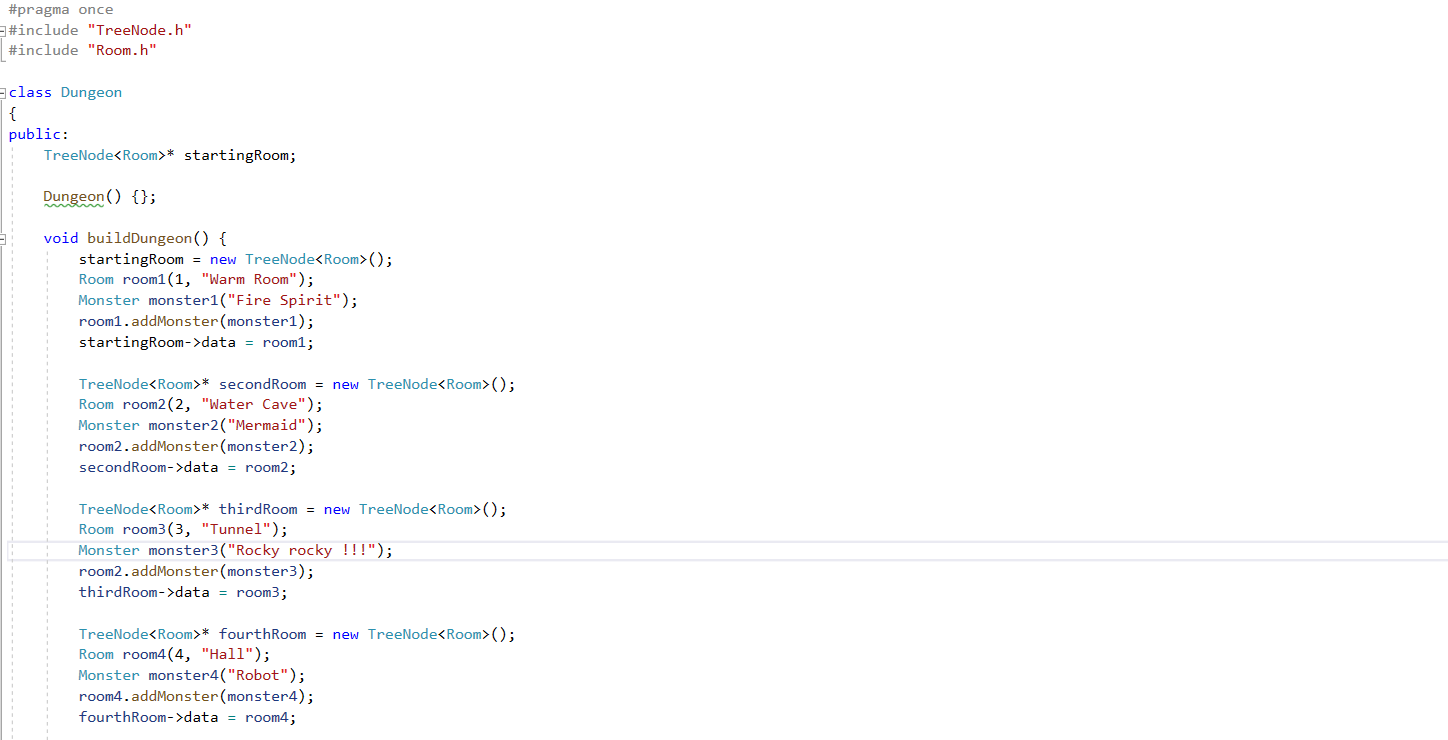
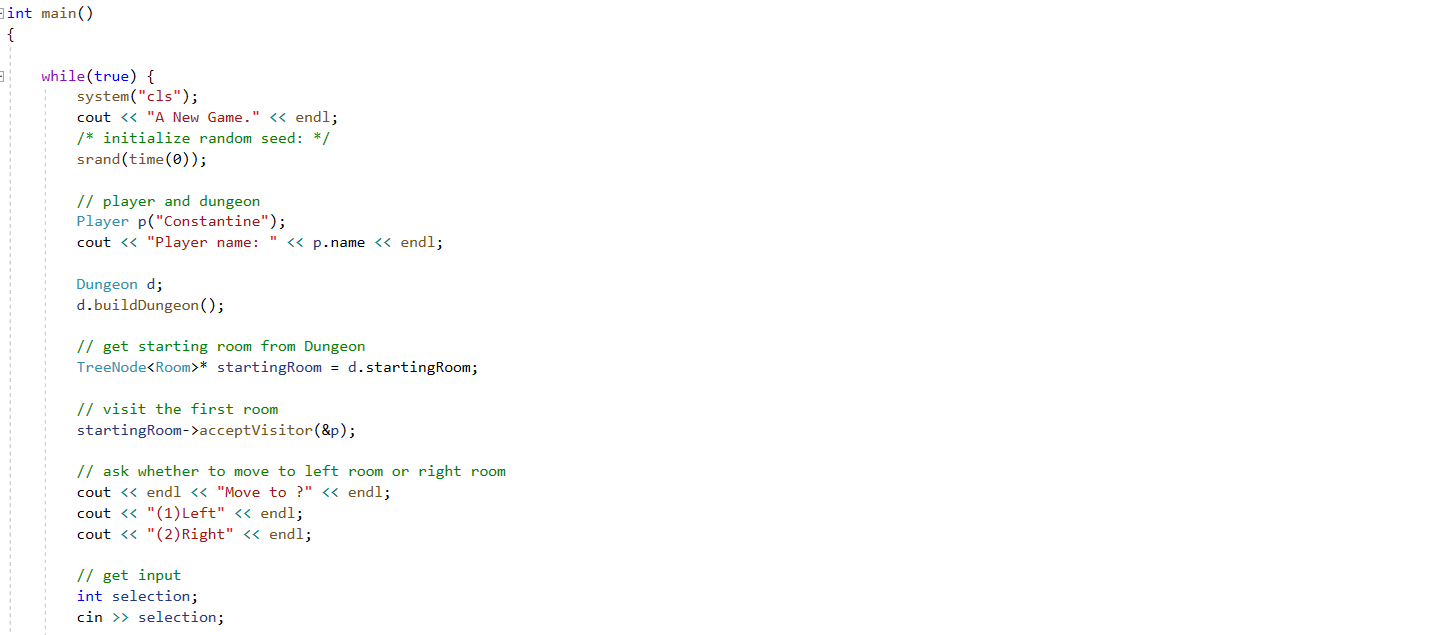




Figure: Dungeon class (Dungeon.h)



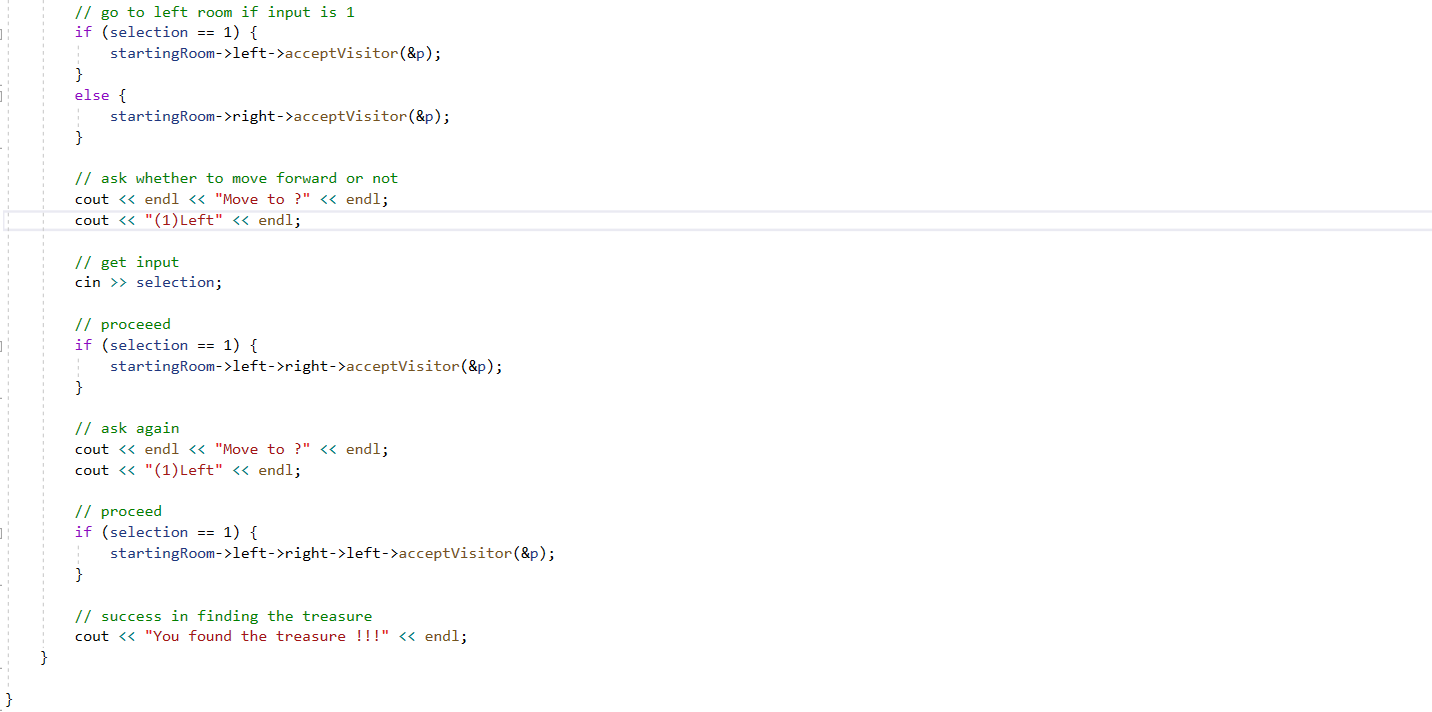


Figure: main (main.cpp)

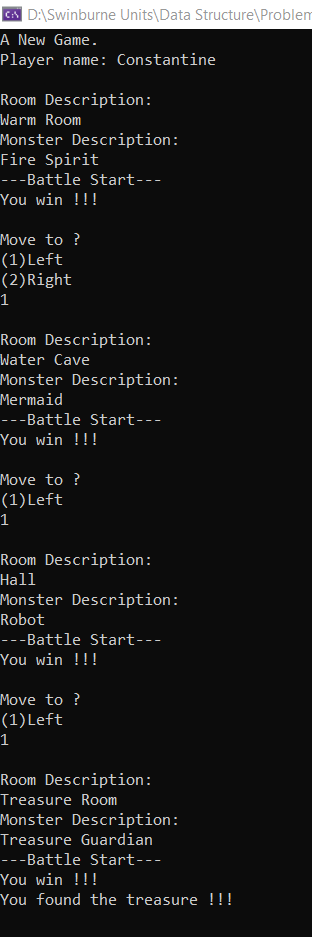


Figure: console output

**Troubleshooting**

The tree built in this task is different than the classic tree. Thus, I could only use the resources (https://stackoverflow.com/questions/931734/tree-datastructures) on the internet as reference and must think by myself. Luckily, the building of tree is hard coded, thus it is easier to debug through call stack.

**Task 4**

**Description**

Research about visitor design pattern and implement the player actions using it.

**Concept**

A way of separating an algorithm from an object on which it operates is what visitor design pattern trying to accomplish. By applying this design pattern, the software gains the ability to add new algorithm to existing object without modifying the object. This is in align with the open/closed principle. Through the visitor, new functions could be added to the family of existing classes, without modification on the classes. The means that the visitor class will be the one that implements all the new functions.

**Code**



Figure: Visitor interface (Visitor.h)

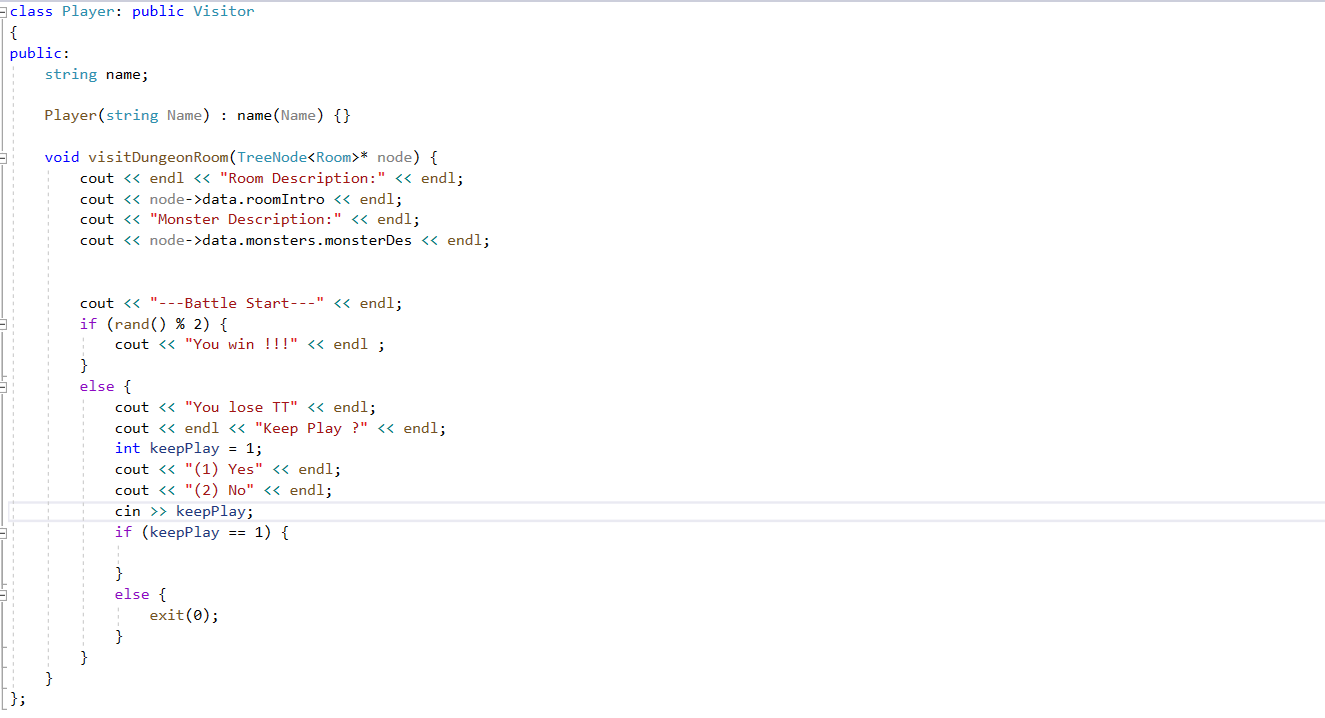


Figure: Player class (Player.h)

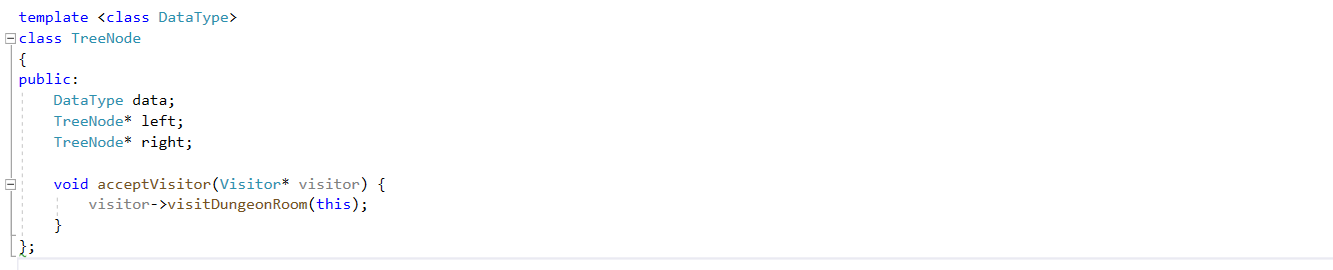
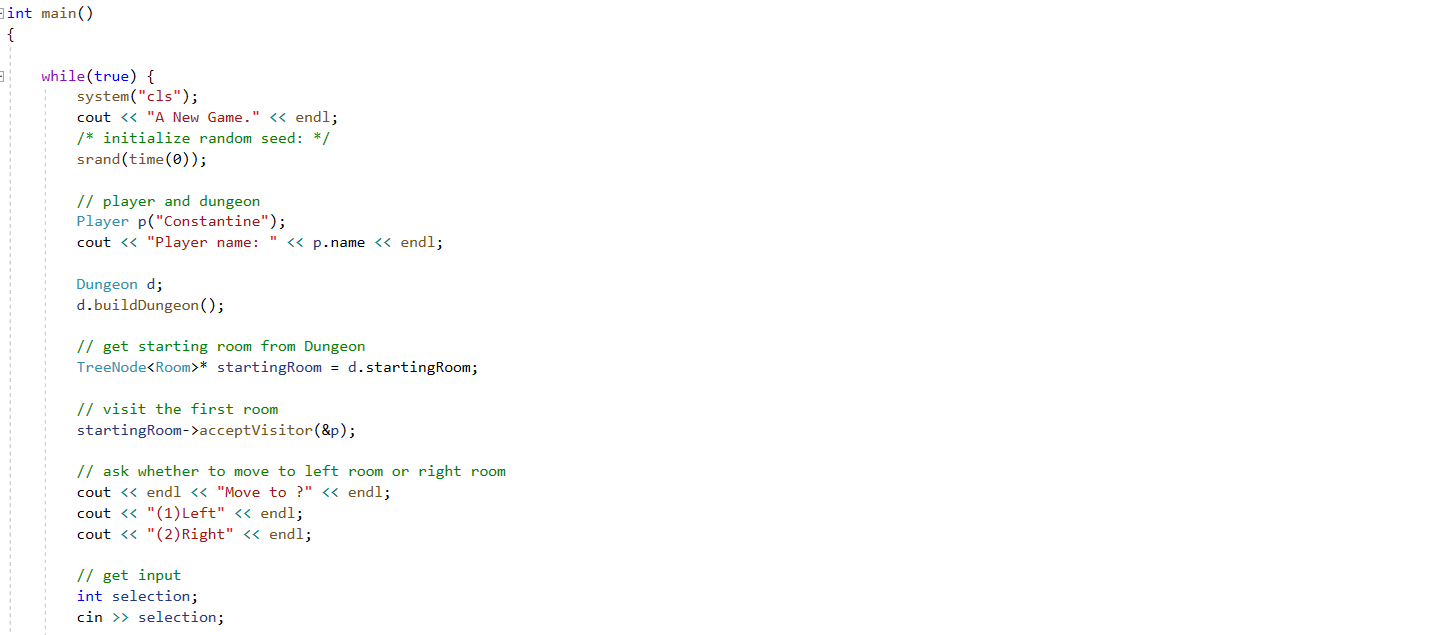


Figure: Tree node class (TreeNode.h)



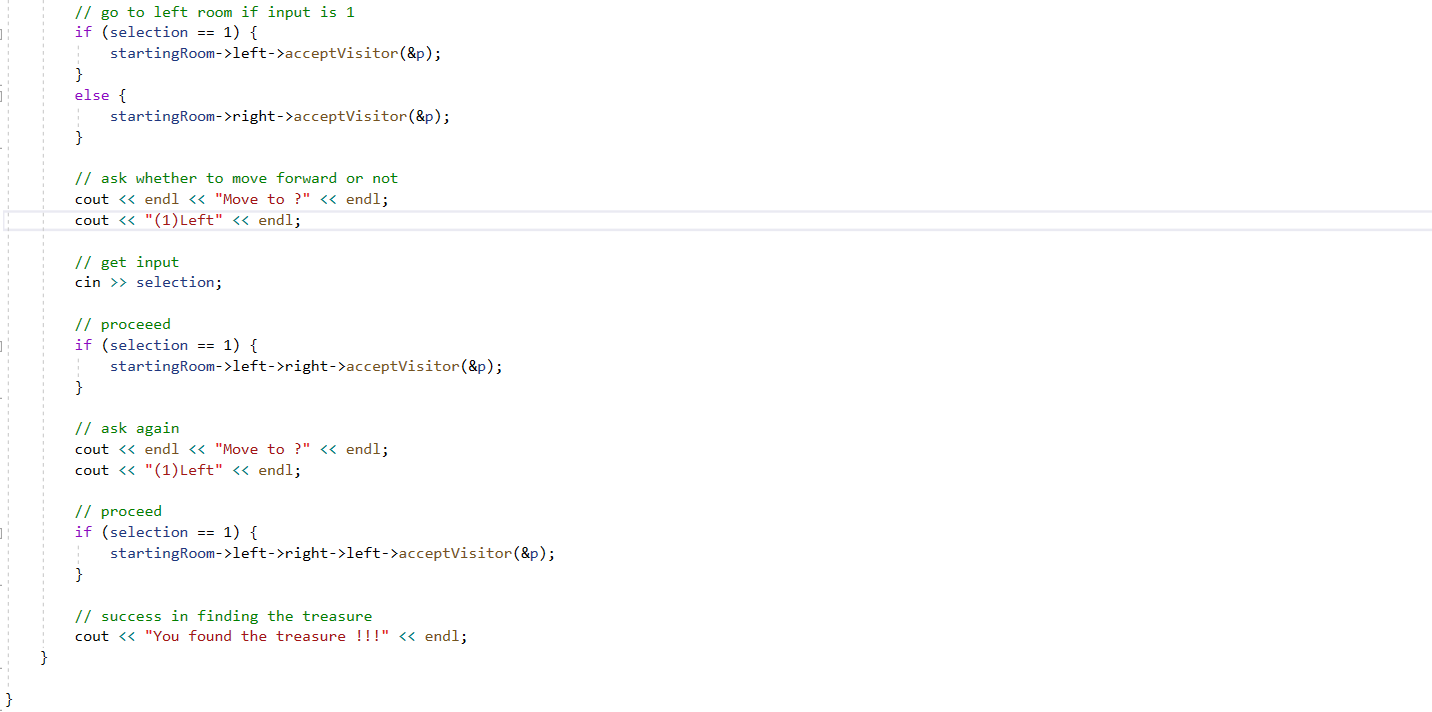


Figure: main (main.cpp)

**Output**

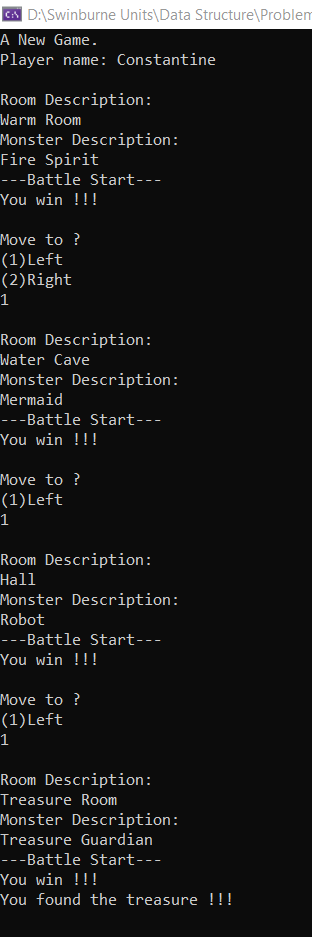


Figure: Console output

**Troubleshooting**

The understanding of visitor pattern is one of hardest task I had encountered in this unit. To be honest, the lecture slides are not helpful at all. Luckily, Dr.Mark provide with a youtube link (https://www.youtube.com/watch?v=TeZqKnC2gvA&feature=youtu.be) that explain visitor pattern in layman term. This Stack Overflow question (<https://softwareengineering.stackexchange.com/questions/333692/understanding-the-need-of-visitor-pattern>) provides great help too.

**Appendix**

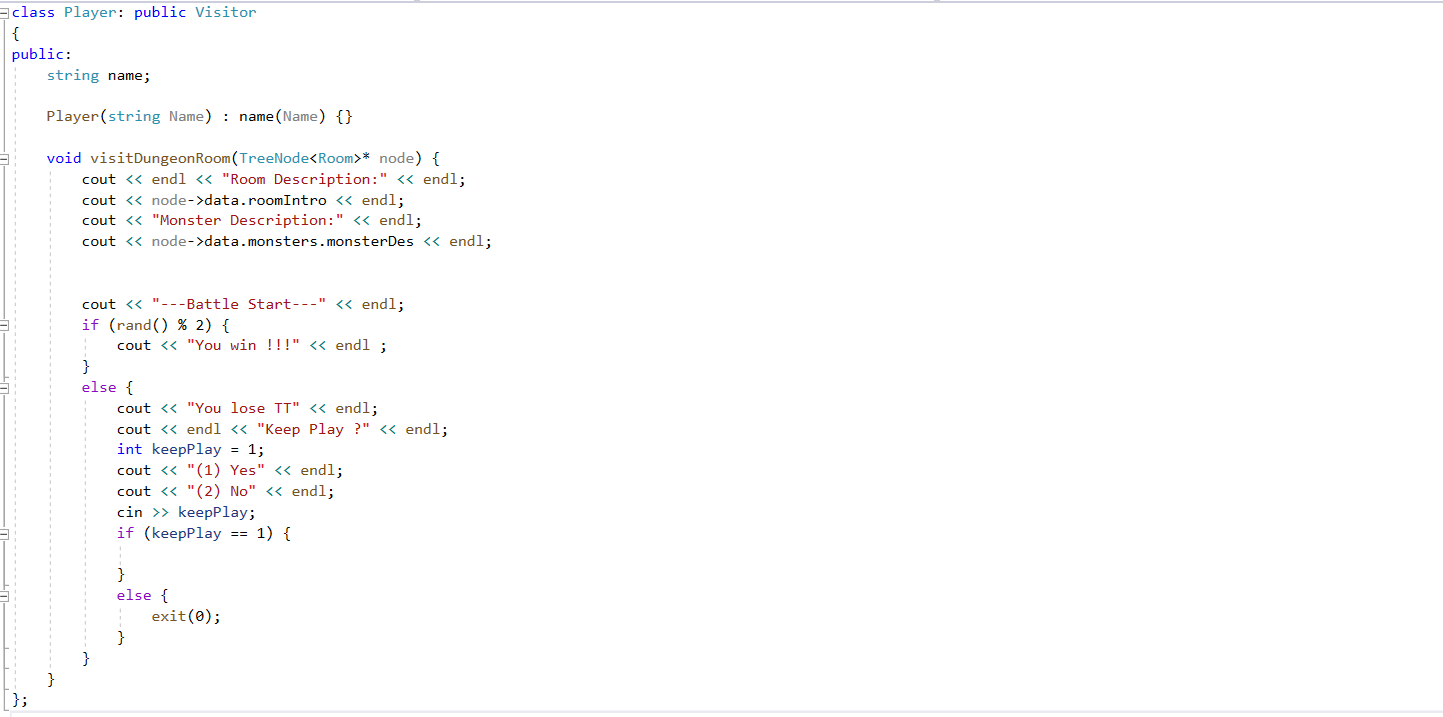


Figure: Player class (Player.h)



Figure: Visitor class (Visitor.h)

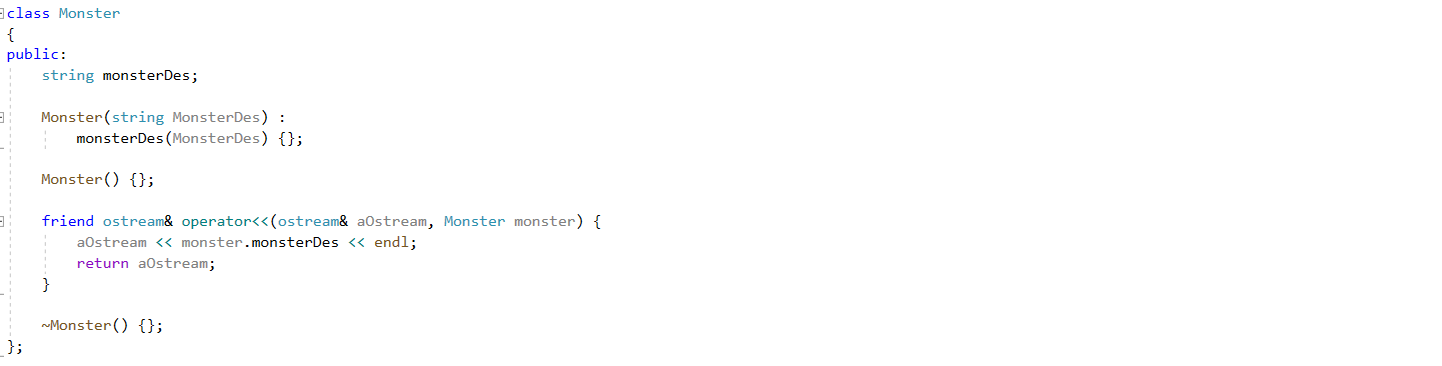


Figure: Monster class (Monster.h)



Figure: Room class (Room.h)

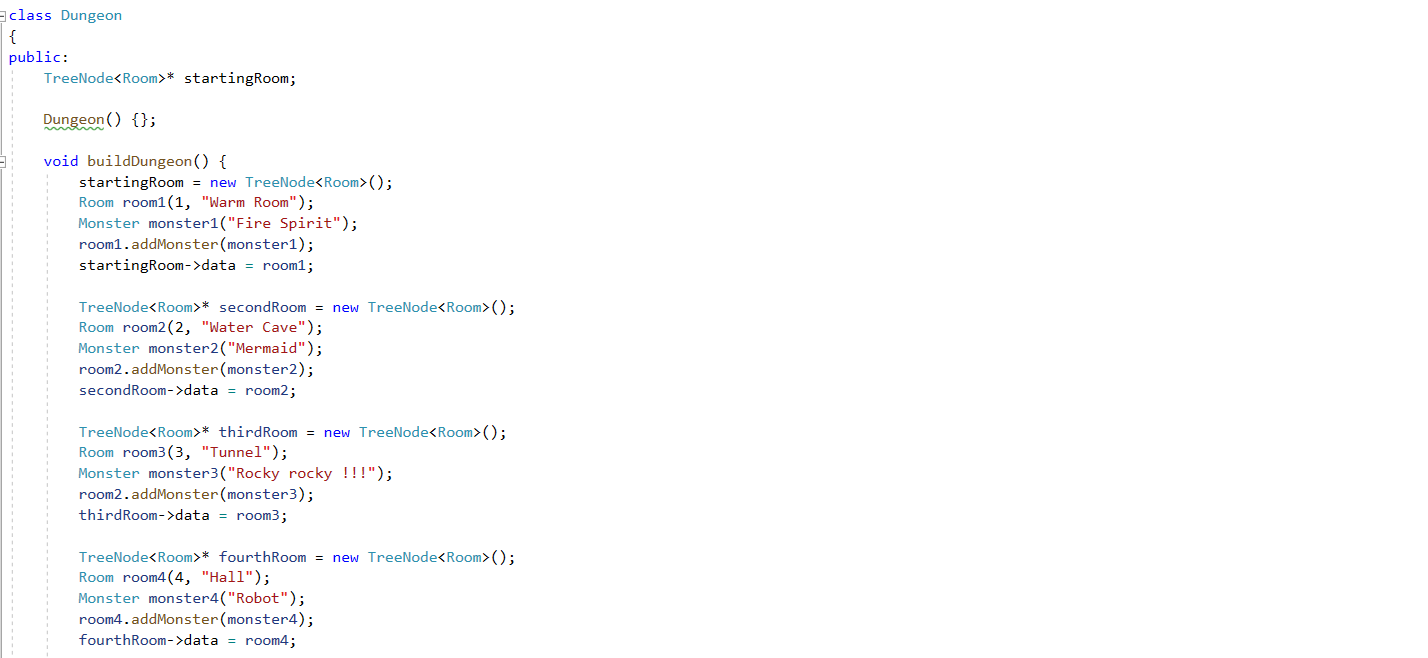




Figure: Dungeon class (Dungeon.h)

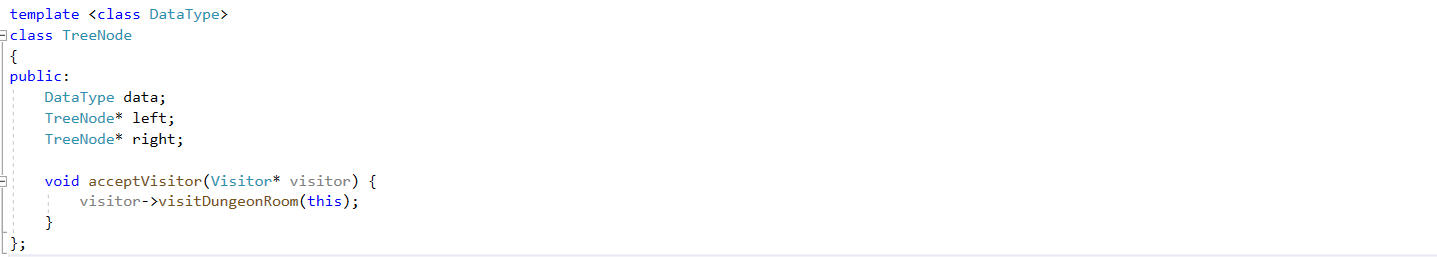
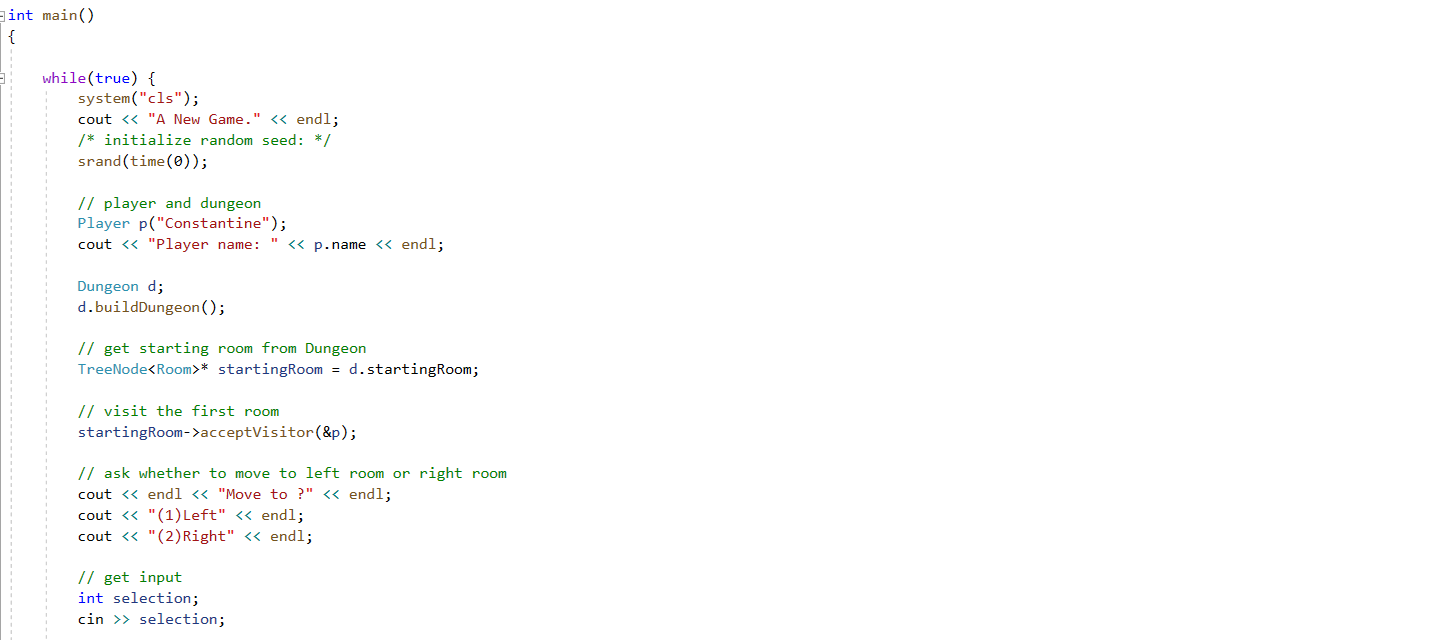
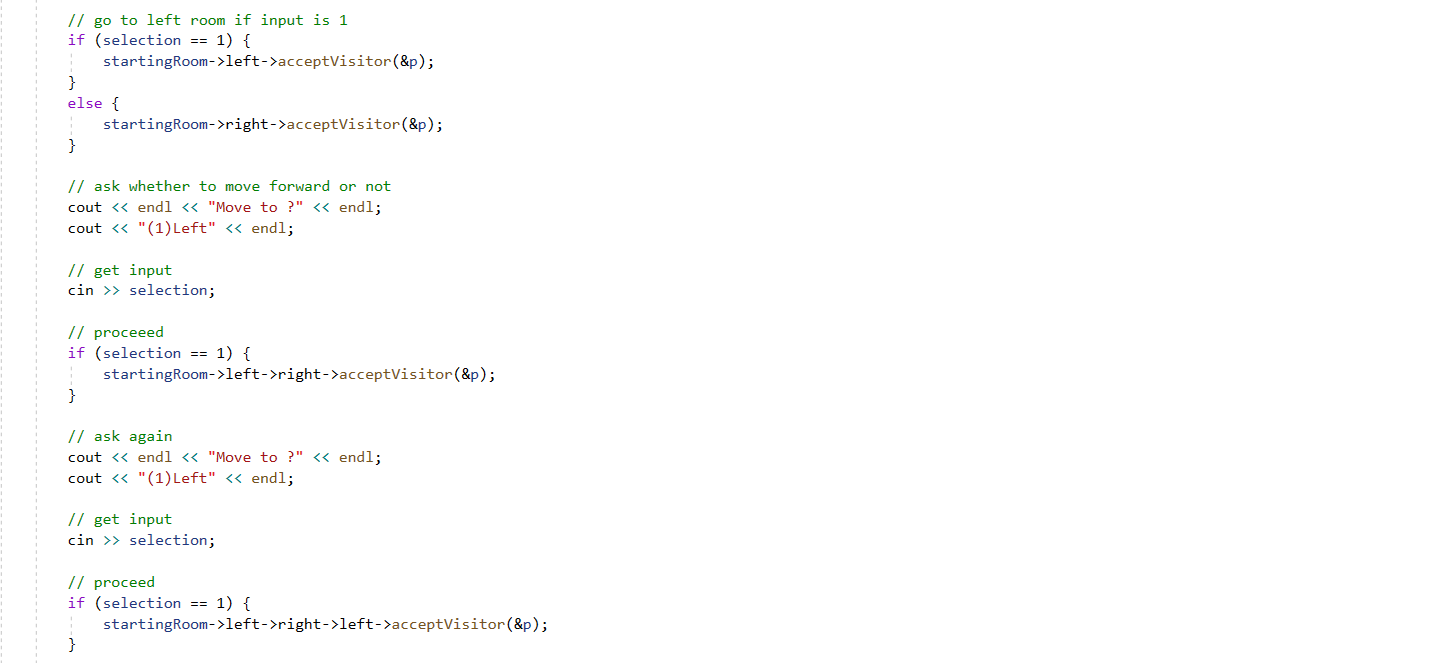


Figure: Tree node class (TreeNode.h)





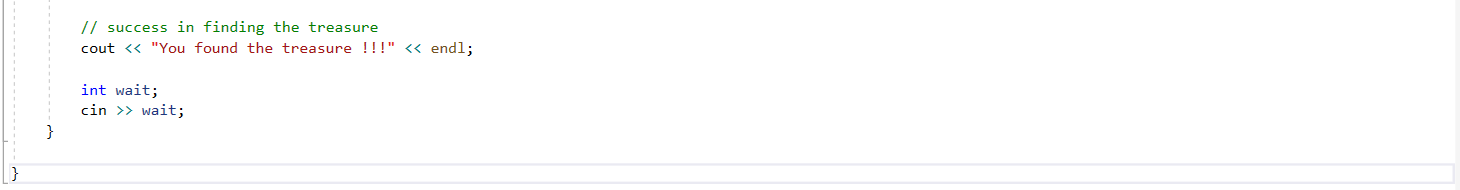


Figure: main.cpp