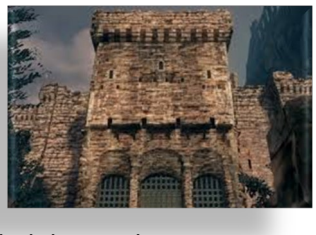
The Scowler's Castle



Description

The period known as the dark ages are called "dark" because very little information was recorded during that time. However, one story did survive by being passed down from generation to generation in the form of a tale—the Tale of the Scowler's Castle.

His name has been lost to history so we will refer to him as the Scowler. The Scowler was an old, crusty, mean fellow who made his fortune the same way that other noblemen did during the dark ages did, by exploiting the weak, poor, and desolate. In the dark ages, business was good for men like these as there were many downtrodden for them to prey upon.

The Scowler of our story was the worst of all. He had become so wealthy that he had built himself a castle. It was not a castle like those of his peers, it was more of a ranch style castle; a one-floor ordeal and in the shape of a square. It looked more like the box that the castle was shipped in than it did a traditional castle. He built such a castle because he was a miserly man.

One day, the hero of our story had his world turned upside-down. He had become deeply indebted to the Scowler despite the urgings of his beautiful young wife. The Scowler decided the proper remuneration of the debt would be our young hero's wife. The Scowler had her abducted and brought to his castle to become his servant. Oddly, even though the hero's wife was young and stunningly beautiful, the Scowler had not considered the debt to be paid in full, so he had our young hero's house burned to the ground as a warning to pay what he owed.

This is where you come in. You were the best friend to our young hero and his wife. You decided to help out by harnessing your awesome powers as a programmer and hacking into the Scowler's security system to provide our young hero with tablet computer that displays a real-time layout of the castle so he can find his wife and save her from abuses that are too foul to be imagined. What, they did not have computers in the dark ages you exclaim. How would you know? There are no records of that period save for this tale passed down from generation to generation.

Task

In the Scowler's castle there are guards. Three types of guard to be exact. The worst-trained guards are those individuals who were forced into servitude by the Scowler. These guards are known as indentured guards. They are a bit lazy and tend to not move around very much and do not really care about protecting the Scowler's property.

The second type of guard was hired by the Scowler but are not paid very well. These guards move around the castle but do so aimlessly. These guards are known as roving guards.

The third type of guard is the most highly trained of them. These guards have dogs and actively search for an intruder. They are paid fairly well by the Scowler and that's why there are so few of them. These guards are known as hunter guards.

Regardless of the quality of guard, if our hero enters a room with a guard in it or a guard enters the room where our hero is, our hero will die. The guards are armed with weapons that our young hero cannot survive. Also, if a hunter-guard enters a room directly adjacent to our hero, our hero will be slain.

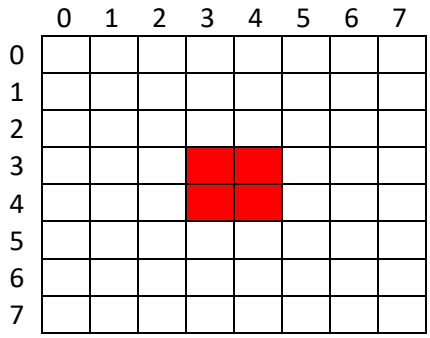
Our young hero's task is to elude the guards, find his wife, and escape. Our hero enters the castle using the northwest servants' door. He knows where his wife is located because you, the C++ expert, who hacked the Scowler's computer-

controlled security system by using a buffer overflow vulnerability. I know, more computers?! Well, if you can write a computer simulator in the dark ages, the Scowler can have a computer-controlled security system.

Activity

Create a program that our young hero will use to save his gorgeous young wife. The program has these requirements:

- The castle has 64 rooms arranged in an 8x8 grid.
- Our young hero always starts his rescue in room (0,0) which has an exterior door. He must find his wife and make it back there alive. This northwest door is the only door for which he has a key.
- Moves can only be made to directly adjacent rooms. Our guards and hero cannot move diagonally.
- The following number and type of guards are randomly located throughout the castle:
    - 5 indentured guards
        - For each move our young hero makes, these guards themselves have a 15% chance of moving.
        - When they move, they move in a random direction.
    - 4 roving guards
        - With every move our young hero makes these guards move as well but in a random direction.
    - 2 hunter guards
        - These guards can sense where our young hero is if they are within two rooms of the young hero. Once detected, with each move of our hero, these guards move towards him. These guards need only be one directly adjacent room away from our hero to find and kill him. These guards can only move to directly adjacent cells.
- Our hero's wife is always located in one of the bedrooms of the castle. She cannot move because she is always tied up. These rooms are located in the center of the castle as shown in red in the diagram below. With the start of each play, she will be placed in a random bedroom and stay there until she is rescued.



- In this castle more than one person can be in a room at the same time. Guards will often enter a room in which another guard is. They may also enter the room of our hero's young darling wife.
- All the Person objects are visible in your program at all times.
- There are 6 actions that our hero can take: move west, move east, move north, move south, pause (don't move one turn), and carry (to pick up his wife), [N, S, W, E, P, C]. Of course, C (carry) uses a turn while our young hero picks up his wife and can only be used when he is in the same room as his wife. If C is entered while our young hero is not in the room with his wife or he is already holding her, then the action will be the same as P (pause). The P option causes the young hero to do nothing for a turn. The other guards continue to move.
- An indentured guard will be represented as an "I". A roving guard is represented by an "R", a hunter guard by an "H", our young hero's wife by a "W". Our young hero by a "Y".

Design

- You must create a Castle class.

  - This class will be the controller for the entire simulation.
  - It will use aggregation to hold the 11 guards in a collection of Guard **pointers**. Use a std::vector for this collection.
  - The Castle class will also hold our young hero and his wife as separate variables of type Innocent **pointers**.
  - The only way the wife can move is if she is carried by the young hero.
  - Make sure the castle has a destructor to delete all the objects created with the new keyword when done.
  - The castle will have a public function called, "move".
  - The castle will also have a friend function that overrides the stream insertion operator (<<). This function will insert into an outstream the layout of the castle with the position of all of the Person objects in it.
- Create a struct called Position which has two public data members, x and y. This will hold the cartesian coordinates of each Person. This struct must have a constructor that takes in the initial values of x and y and a copy constructor that takes a position as an argument.
- Create a person class that will be the base class for all the guards, the young hero, and his wife **and should be abstract**.
  - This class will have a protected Position data member which represents the Person's initial or current position.
  - It will also hold the type of person in the form of a protected char.
  - This class also has a constructor that take a Position and type as its only arguments. This will receive the initial position of the of the person and the type of person [I, R, H, W, Y]. Override the equality and inequality operators, == and !=. These compare the position of two Person objects.
  - This class also has a pure virtual function called move which takes the move as its argument which has a default value of 'A' for auto.
- Create a guard class that derives the person class.
  - This class will have a pure virtual function called move which has a move as its parameter which defaults to 'A' for auto.
- Create the three guard classes, IndenturedGuard, RovingGuard, and HunterGuard which derive the Guard class.
  - These three specialized versions of the Guard class will override the *move* function in Guard.
  - Each specialized overridden *move* function ignores its position parameter and calculates its own move or decides not to move at all as in the case of the IndenturedGuard. These moves must be legal moves.
  - Their constructors take as a parameter their initial position.
  - The constructor for the HunterGuard class also takes a second parameter which is the pointer to the hero object. This way the hunters know where the hero is at all times and can hunt him down.
- Create an Innocent class that inherits Person. This is the class that represents our hero and his wife.
  - This class overrides the pure virtual *move* function in its inherited Person class. The move function must accept only a valid Position to which to move. It should throw a runtime exception if the move is invalid.
- Write a program called Scowlers_castle.cpp which handles all user input and output as well as turn taking and the player's move making.
  - When the main program needs to output the state of the Castle object (we'll assume it's called "castle" for this explanation) it will be inserted into the cout stream like so, std::cout << castle << std::endl.

- The program should prompt the user for a valid move and re-prompt if an invalid move is entered. Valid moves are listed above but restricted by the young hero's position in the castle. In other words, if our hero is at (0,0) he can only move east or south.
- The hero saves his awesomely beautiful young with if their positions are at 0,0.

- The headers for all classes will be included with this project but might be incomplete in terms of inheritance and the virtuality of functions.
  - You may have to declare the inheritance and make functions virtual or pure virtual as needed.
  - All objects will be created on the heap so the destructor in the object or program that created an object is responsible to delete it.
  - You must declare include-guards for all of the classes and the struct.
  - You must write the classes definition in a separate cpp file.

Design Notes

- Stay true to the requirements of this project. This is meant to gain a better understanding of inheritance and polymorphism.
- You can create private helper (utility) functions in any of your classes but do not add any public functions.
- Create any and all objects with the exception of container objects on the heap. Use the new keyword for this but be sure delete them from the destructor of the object responsible for them.
- Always try to code for efficiency.
- The start of play should look like this:

```
Welcome to the Scowler's Castle
---------------------------------
|   |   |   |I  |   |   |   |   |
| Y |   |   |   |   |   |   |   |
---------------------------------
|   |I  |   |   |IR | R |   |   |
|   |   |   |   |   |   |   |   |
---------------------------------
|   |   |   |  W|   |   |   |   |
|   |   |   |   |   |   |   |   |
---------------------------------
|   |I  |   |   |   |H  |   |   |
|   |   |   |   |   |   |   |   |
---------------------------------
|   |   |R  |   |   |   |   |   |
|   |   |   |   |   |   |   |   |
---------------------------------
|   |   |   |H  |   |   |   |   |
|   |   |   |   |   |   |   |   |
---------------------------------
|   |   |I  |   |   |   |   |   |
|   |   |   |   |   |   |   |   |
---------------------------------
|   |   |   |   |   |   | R |   |
|   |   |   |   |   |   |   |   |
---------------------------------

Your move [N,S,W,E,P,C]:
```

- Note, if there are more than one of any person in a room, only display one letter for each type this way you can fix the position of each type in the output.

Requirements

- **Do not change the user interface or the behavior outlined in these specifications.**
- **Add pre and post condition comments to function prototypes in each header file.**
- **Create a separate definition file for each header file.**
- **You can add private members to the header file of any class but do not change the public or protected sections unless you get permission.**

Reminder

You are responsible to do your own work. **This is not a team project**. **Do not show your code to anyone and do not look at, or copy any code from any source (even the lecture notes or the book); create your own code.** Our lectures and the book contain all the information you need to complete this project. Any violation of the school's academic integrity policy or the policy of this class will result in a zero grade and an academic misconduct report filed with the school; no excuse will be accepted.

Your program must compile and run to receive any credit. If I cannot compile your program, you will receive a zero for your score. Treat this like any other exam, start right away and put effort into it.

Rubric

This project/exam is worth 300 points. The points will be distributed according to the chart below. If the program does not compile or is missing, then no points will be awarded.

Partial credit will be awarded for the features of the program that do work if the program compiles and runs.

Program 300
Requirements 120
Correctness 140
Code Quality 40

Note: “Code Quality” refers to following good coding practices. Indentation, spacing, proper naming of identifiers, and modularization (other than that specified in the requirements) make up this part of the grade.

What to Submit

- All of your header and source code files. They must compile and run in VSCode to get credit.
- Hint: the sooner you get started the more you will enjoy this project and the better you will do. No late projects will be accepted. I will offer help and hints if you start early but please do not ask me to look at your code.