The C++ Programming Language

1 Deadline

Monday 29th of April at 17:00
Feedback will be released on Monday the 20th of May (3 wks later).

2 Weighting

50%

3 Overview

You will create a small interactive console-based (i.e. text based) adventure game. The project consists of 9 tasks. You are expected to use Visual Studio 2022 Community Edition to build your game. You should complete the tasks in order, as some tasks assume that previous tasks have been done. Each task builds upon the previous one. Do as much as you can and then package your project for submission. Your solution should demonstrate your knowledge of C++. Work alone: You must work alone on the assignment. Plagiarism checks will be run. If you notice any typos or mistakes in the assignment that I may have missed, or if you feel something is unclear, please contact me (andyrox@liverpool.ac.uk) or post a question in the Canvas COMP282 Discussion page (there is a specific discussion relating to clarification of the assignments).

4 Submission Requirements

I will upload a separate document to Canvas describing submission requirements but it is likely that you will upload your cpp and h files to Codegrade. I will also upload a rubric describing the mark scheme. Marking will be a combination of running automatic tests and manually reviewing the code.

4.1 Overview: Fantasy Quest Adventure Game

The assignment is to create a basic command-line game using C++ and organise game structure using OOP methods.
The game has a fantasy theme, in a style similar to Dungeons and Dragons, consisting

of a player that can navigate a series of connected locations (using commands `North`, `South`, `East`, `West` or `N`,`S`,`E`,`W`). The player can collect items and engage in combat with monsters. The player's score in the game increases when they defeat a monster. The objective of the game is to achieve a high score (by defeating monsters) and to defeat the 'end of level boss'. There should be some introductory text at the start of the game to set the scene, and the player can leave the game at any point by typing `quit`. The player can type `inventory` or `inv` to see a list of all the items they hold. The player can type `fight` to engage in combat with a monster, or type `collect` to collect all items in the current room.

You must use the specified class, method and variable names to ensure your code can be tested. You will get marks for using STL containers and iterators to store and manipulate the various objects in the game. You are free to add further methods/ variables/ functions as you wish. Be sure to add comments your code, briefly explaining the purpose and functionality of classes, methods, and significant blocks of code. Also remember to implement error handling. In the descriptions of programming elements below, I have deliberately not specified function parameters and return types - this is for you to specify. Similarly I have not specified where pointer use would be appropriate.
I have specified most of the monsters/locations/objects - this is so that the gameplay can be tested.

The assignment is organised into tasks.

4.2 Task 1 - create game classes

The following classes should be written:

- **Character** The Character class defines characters in the game. It should have the following methods: **setName**, **getName** which will get or set the name of the character, and **getHitPoints** and **setHitPoints**, which get or set the hitpoints, or 'health rating' of the character.
- **Monster** Monster should be derived from Character.
- **Player** Player should be derived from Character. It should have a member variable **score** (an integer that keeps track of the player's score in the game), which is set and read using **setScore** and **getScore**, and three containers **weapons**, **potions** and **treasures** which hold the player's items. It should also have the private member variable **currentLocation** which represents the player's current location. This can be set using **setCurrentLocation** and read using **getCurrentLocation**.
- **Item** Item should have the following methods: **getName**, **setName**, which will set or get the name of the item.
- **Potion** Potion should be derived from Item. It should have a member variable **strength** (integer), which should be accessible via **getStrength** and **setStrength**.
- **Weapon** Weapon should be derived from Item. It should have a member variable **power** (integer), which should be accessible via **getPower** and **setPower**.

- **Treasure** Treasure should be derived from Item. It should have a member variable **value** (integer), which should be accessible via **getValue** and **setValue**.

- **Location** A Location object represents a place within the game. It should hold data including **name** and **description** (accessible via member functions **setName**, **getName**, **setDescription**, **getDescription**). It should be possible for the programmer to populate a location object with other game entities, using the following member functions: **addExit** (creates an exit, either N, S, E or W, leading to another location object), **addMonster** (puts a specific monster at this location), **delMonster** (deletes monster), **getExits** (displays a list of exits to the player), **addItem** (places a specific item at this location). Function overloading should be used on the **addItem** method so that **addItem** handles adding potions, treasures or weapons to the location. Member variables should include the containers **weapons**, **potions** and **treasures** which hold the player's items. And **exits**, which holds the exits leading to other location objects.

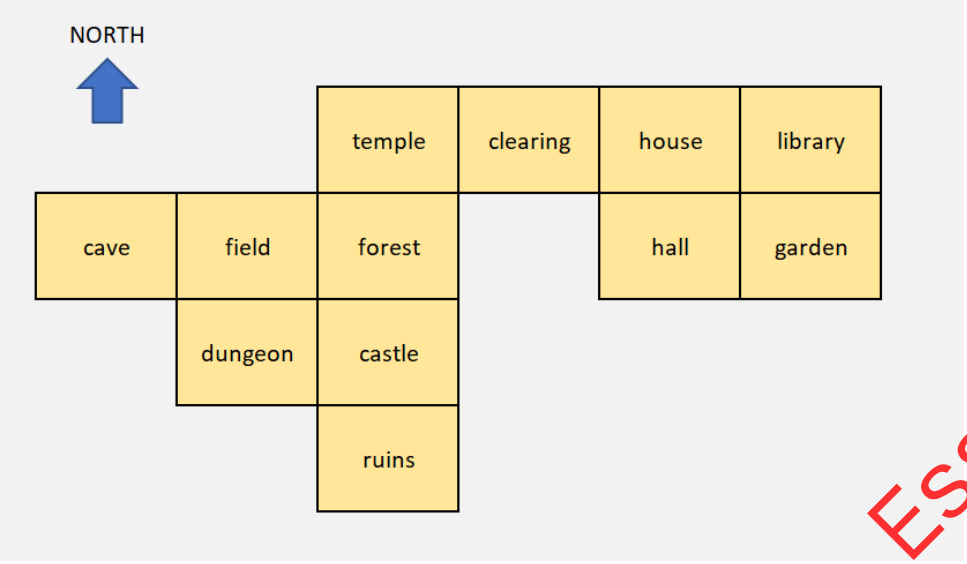
4.2.1 File Structure

The File Structure should be according to good C++ practice, with declarations in the .h files and implementations in the .cpp files. However for simple one or two line functions, those implementations can be kept in the .h file, allowing the compiler to inline them and reduce function call overhead. For this assignment we will also keep derived classes in the same files as their parent classes. AdventureGame.cpp should have the main() function. Files are:

- Character.cpp
- Character.h
- Item.cpp
- Item.h
- Location.cpp
- Location.h
- AdventureGame.cpp

4.3 Task 2 - Build the game world

Build the game by creating location objects and linking them together using the objects' **exits** member variable. Create 12 location objects that are linked to each other according to the map shown.
Here are the location names:
cave, temple, dungeon, castle, clearing, hall, garden, library, forest, house, ruins, field.



Set each location object's description (you can put whatever you want here) and a name to be displayed after "you are" - e.g. 'ruins' might be given the name 'in some ruins'. After building the game, set the player's current location to the **clearing** - that will be the start location of the game.
Create a game loop in the **main()** function. At this stage the game loop should simply get a command from the player, and then describe the location again. If the user types `quit`, the game should display the user's score (which at this stage will be 0), and end.

4.4 Task 3 - Room Navigation

Modify the game loop so that it now tells the user where they are, and what exits are available (N, S, E or W).
If the user types a direction (`N, S, E` or `W`, or `North, South, East` or `West`, or `n,s,e or w`), the game should work out if an exit exists in that direction, and if it does, the current location should change to that location.
If an exit does not exist, the game should tell the user this, and remain at the current location.

By the end of this task the user should be able to navigate around the game, and quit the game if necessary.

4.5 Task 4 - Adding Items

Create some items (see below).
Treasures: bag of coins, treasure chest, emerald, ruby, sapphire, diamond, gold ring.
Potions: Blue Healing Potion, Purple Healing Potion, Red Healing Potion.
Weapons: dagger, sword, crossbow, club, stick.
Use **addItem** to add these items (weapons, potions, treasures) to the locations in the game as follows:

	treasure	value	potion	strength	weapon	power
cave	emerald	40				
field	sapphire	40			dagger	5
dungeon			red healing	40		
temple	diamond	60			sword	6
forest						
castle	gold ring	60			stick	1
ruins			purple healing	30		
clearing					club	3
house	treasure chest	200				
house						
hall	bag of coins	50	blue healing	20		
library						
garden	ruby	10			crossbow	10

Change the game loop so that it announces what items are in the location, if any. This information should be obtainable from the player's current location.

If the user types `collect`, all of the items in the location should be collected by the user. If the user types `inv` or `inventory` they should see a list of all their items, in the three groups (Potions, Treasures, Weapons). Sort the lists alphabetically. If the user types `drink`, all the healing potions in the user's inventory should be deleted, and their 'strength' points should be added to the player's HitPoints.
By the end of this task the user should be able to collect all items from rooms, drink the healing potions, and list the items in the inventory.

4.6 Task 5 - Adding Monsters

Use **addMonster** to add monsters to the rooms.
Here are the monster names:
vampire, zombie, dragon, goblin, spectre, banshee, ghoul, orc.
Arrange the monsters in the game as follows, with the following hitpoint values:

	monster	hitpoints
cave		
field	goblin	10
dungeon	zombie	8
temple		
forest	banshee	7
castle	vampire	9
ruins	dragon	15
clearing		
house	orc	12
house	spectre	5
hall		
library		
garden	ghoul	10

Change the game loop so that it announces the presence of a monster in a room. Note that the **house** has two monsters.
By the end of this task there should be monsters in locations.

4.7 Task 6 - Combat

By the end of this task it should be possible to engage in combat with the monsters/enemies according to the rules of combat, by typing `fight`.
Create a function called **rollDice** which will return a random integer from 1-6.
Create a member function of the Player class, called **combat**, that will conduct combat with between the player and any enemies in the current location. The rules of combat are as follows:

1. Both *Monster* and *Player* have a hitpoint value. Once a character's hitpoint value reaches 0, that character is defeated.
2. The *Monster* strikes first.
3. Their damage to the player is determined by calling **rollDice** and multiplying the result by 2. (So result can be 0-12).
4. The player's hitpoint total is reduced by the damage inflicted by the *Monster*.
5. The player then strikes back - **rollDice** is called, and this is modified by the weapon with the highest power rating. So if the player has a sword with a power of 10, 10 is added to the dice roll to determine damage done to the monster.
6. This cycle continues until either the player or the enemy reaches 0 hitpoints.
7. After each round, the hitpoints of each combatant are displayed to the user.
8. If there is more than one monster in the room, only one monster is engaged in battle when the user types `fight` - the user needs to type it again to fight the other monster. The monster order should be strongest first - so the player fights the monster with the higher hitpoints first.
9. The *Character* class should have the member function **takeHit** added to it, which will decrease the character's hitpoints by the given amount.
10. If the player is defeated, the player's score is shown and the game ends.

4.8 Task 7 - The Boss

Create a new Player character called 'boss', give it a name and description, and add the boss to the game. **Put the boss in the cave**. Modify the game loop so that when the boss is defeated, the game ends.

4.9 Task 8 - Game Enhancements

Add further items to the game:
Treasures: Cup, Pearl, Key, Book
Potions: Green Healing Potion
You can decide their value, and strength in the case of the healing potion.
Put these items into the possession of some of the monsters. **Make sure that the ghoul**

has the key.
Modify the game so that when a monster is defeated, their items become available in the room for the player to collect.

4.10 Task 9 - Implementing Armour

Create a new child class of Item called Armour.
Add at least 5 examples of armour into the game. Some examples:
ringmail, chainmail, shield, breastplate, helmet, gauntlet.
As well as name and description, the Armour class should have a member variable called **protection**. You can decide values for this. But **make sure that you put the gauntlet, with a protection of 2, in the library.** Armour reduces the effect of damage inflicted on the character. All Character objects should have the ability to use armour, so it should be implemented at the Character (parent class) level. All armour items should be listed to the user when they type `inv` or `inventory`. Modify the combat rules so that during combat, damage inflicted on the player by an enemy is reduced by a random amount between 0 and the total value of protection of all the armour in the player's possession.

5 Example Gameplay

```
Welcome to the Adventure Game.The aim is to defeat the Boss, but you
score points for defeating monsters along the way.
You can use the commands:
n, s, e, w, inventory, drink, collect, fight, quit.

---
You are in a clearing.
It is a still evening.
Exits are E, W.
There is a club here.
>collect
Collecting items.
You collected the club.
---
You are in a clearing.
It is a still evening.
Exits are E, W.
>inv
You have the following items:
==================================
 Score: 0 Health: 25
 Total Armour Protection: 0
Weapons:
 club (+3)
Treasures:
 none.
Potions:
 none.
Armour:
 none.
==================================
---
You are in a clearing.
It is a still evening.
Exits are E, W.
>
```

```
You are in a library.
It is dark in here.
Exits are S, W.
>s
---
You are in a garden.
It is very pretty.
Exits are N, W.
There is a ruby here.
the ghoul is here.
There is a crossbow here.
>collect
Collecting items.
You collected the ruby.
You collected the crossbow.
---
You are in a garden.
It is very pretty.
Exits are N, W.
the ghoul is here.
>fight
Engaging the ghoul in combat...
 the ghoul (10)
Hero vs the ghoul
- - - - - - - - - -
Round 1
19   4
- - - - - - - - - -
Round 2
13   -2
You have beaten the ghoul.
---
You are in a garden.
It is very pretty.
Exits are N, W.
>
```