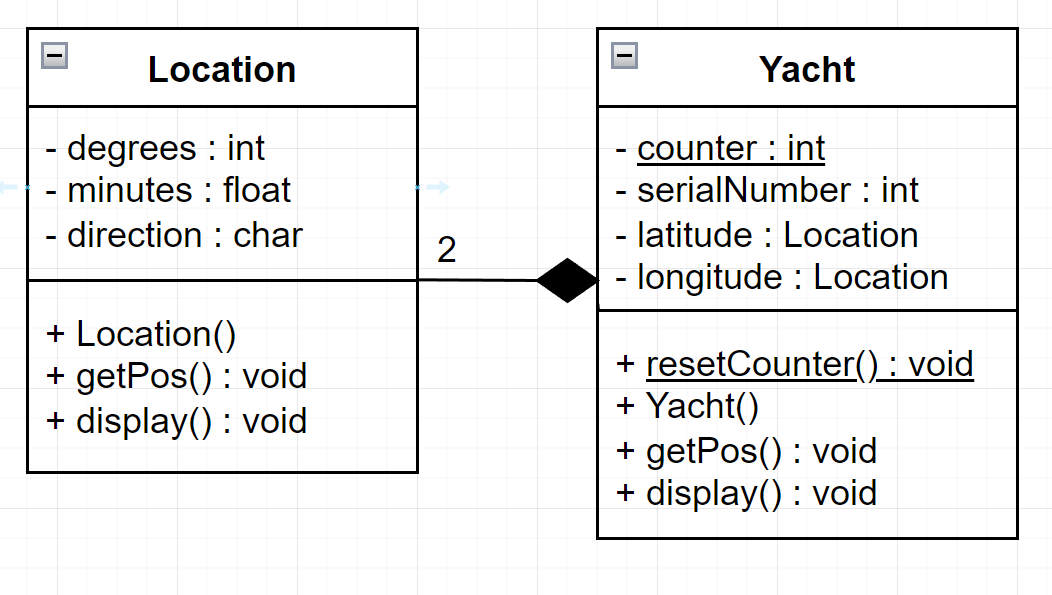
UML Diagrams

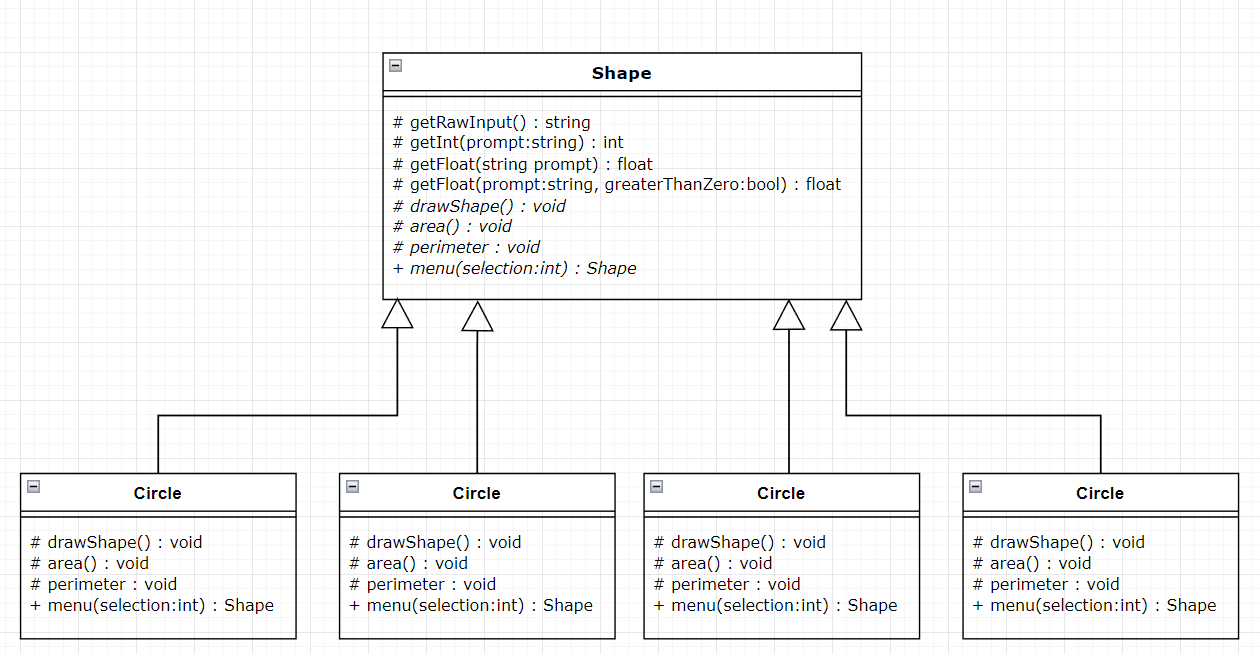
Task 1

Prepare the UML class diagram for the Location class:

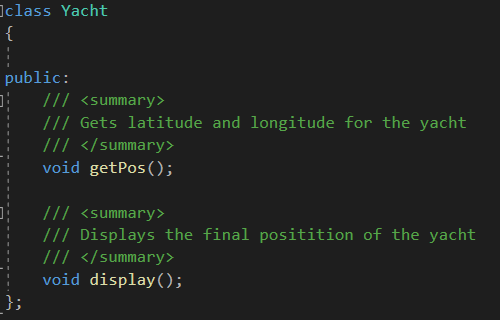


Task 3

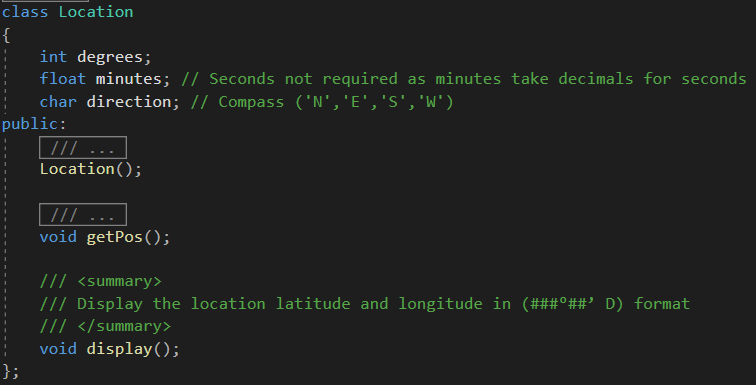
Draw the class diagram UML clearly depicting the access specifiers, inheritance, data members and member functions of all classes planned to use



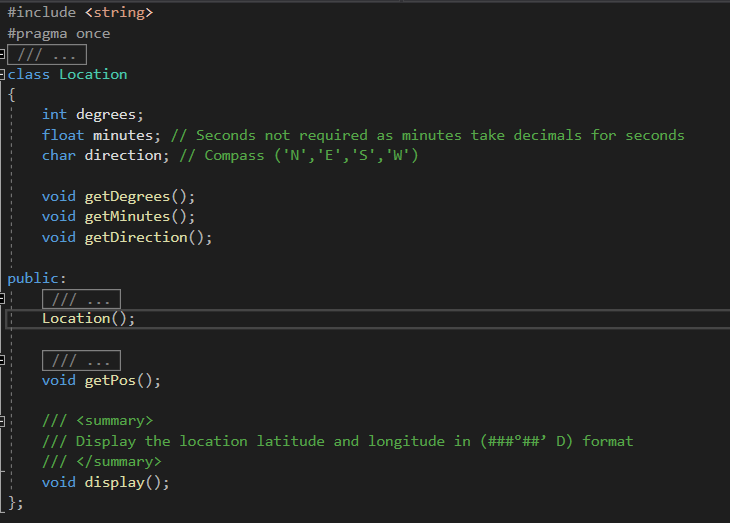
Describe The Process – Task 1

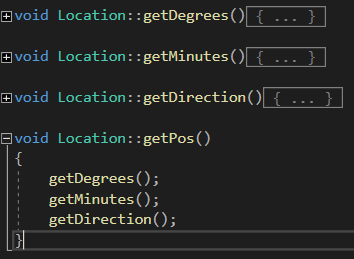
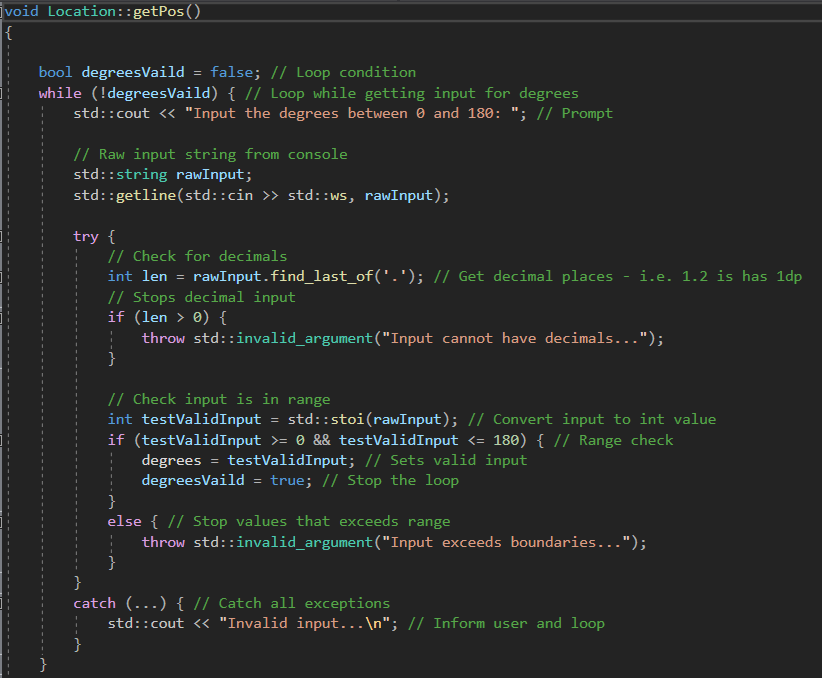
To start off, I decided to make a Yacht class as outlined in brief. So, I added the first two functions outlined in the brief: getPos() and display().

When I was going to code these, I realized that they both used functionality that was going to be made in the Location class outlined in the brief. So, I changed my focus and went straight into making a location class to allow my Yacht class functions to be able to be coded.

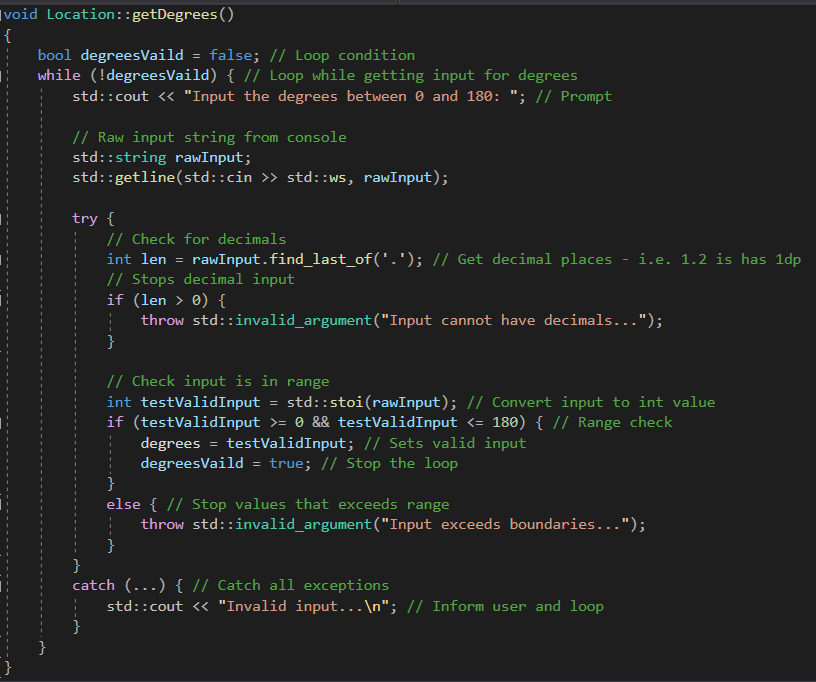


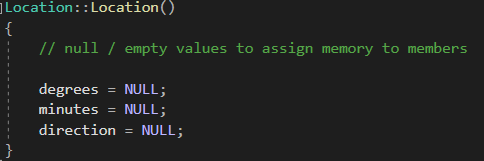
The header file shows that the location class contains all the required members. The degrees, minutes, and direction were all private as they weren’t needed outside the Location class. They were going to used to store either a Latitude or Longitude which have the same concept but are used together to accurate define a location which is how naval vessels, like a yacht, can navigate at sea. I decided I should create the functionality for these functions in the Location.cpp file next so I know how they will be used in the Yacht class.

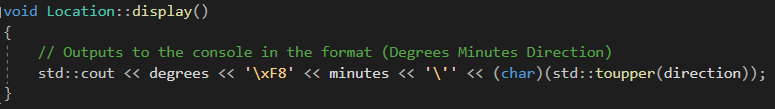


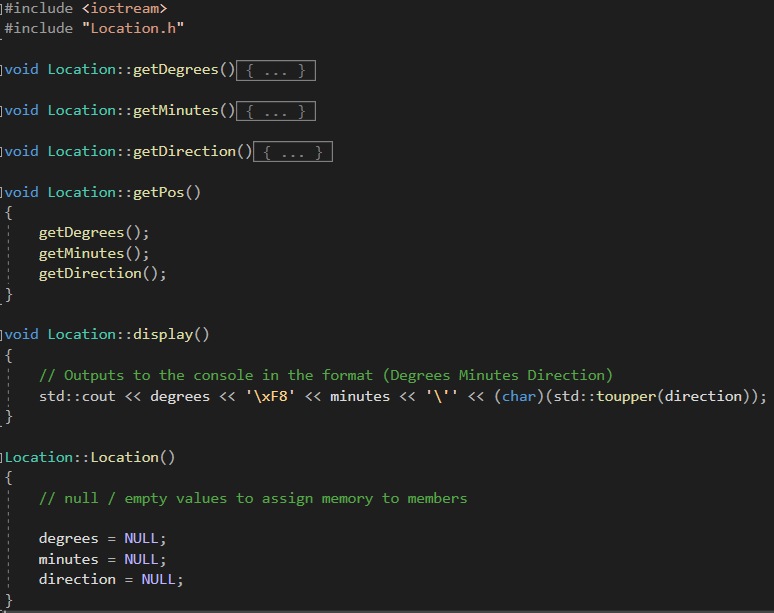
  
When I started coding getPos, I realized that the function was getting long and hard to make changes. So, I split it up into three private functions which get each part of the Location.

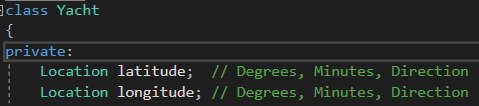
This made the getPos function easy to understand and make edits to each functions algorithm to get each type easier.

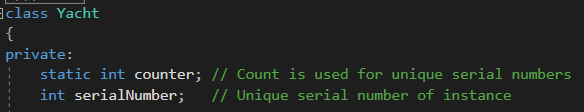
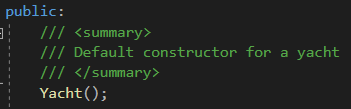
  
I will only show one of the 3 functions because they are similar, just have a different input type / requirement. These functions use a while loop and a try catch to ensure user input into the console is within an acceptable range, doesn’t have decimals if value is an int, and is a valid number if a float or int. This also stops errors occurring if there are conversion errors between string and number. After I completed the user input for getting data in for the location, I made a constructor which set the values into NULL to allocate memory when an instance of Location was created.

Afterwards I added to console output in the formatted way required. This then completed the location class, and I was ready to make my way back to creating the Yacht class.

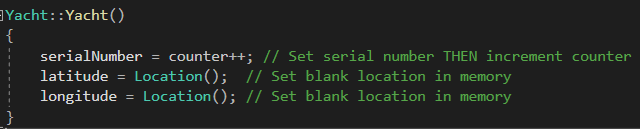


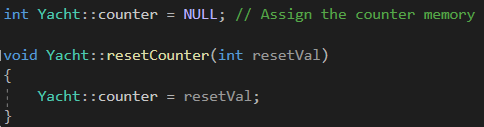
Completed Location class.

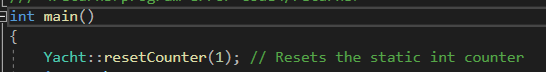
I came back to the Yacht header and added two location members to the class: Latitude and Longitude. While I was in the header file, I also added the serial Number of the Yacht and a static counter variable into the class header.

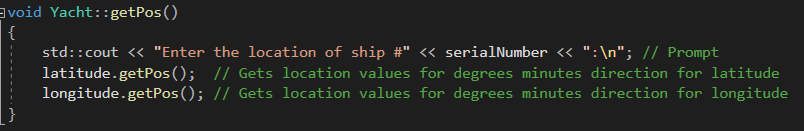


I almost thought I was ready to start coding the functions in the Yacht class but when I was about to start, I realized I needed a constructor for the Yacht class. The reason for this was because I needed a way to assign the serial number and increment the static counter variable.

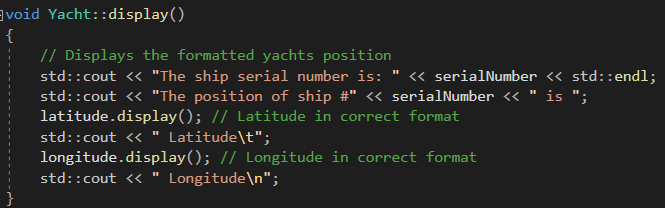


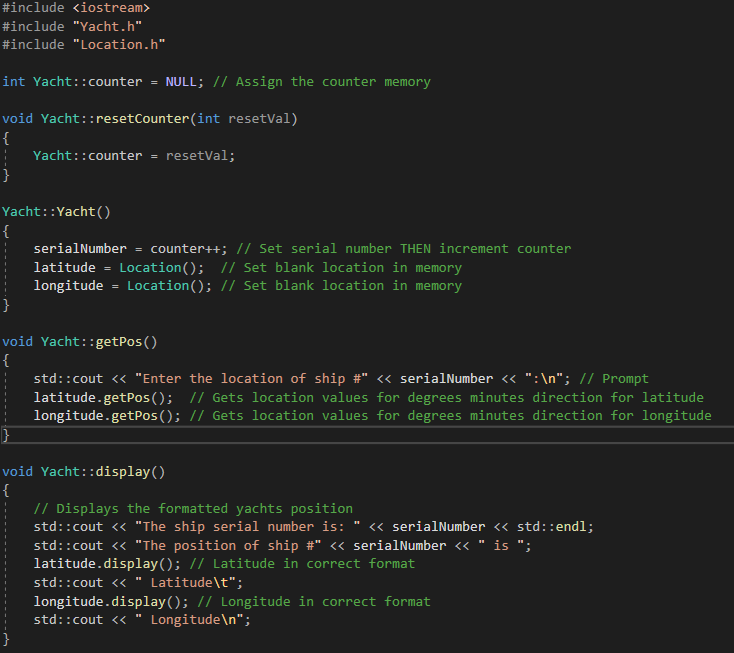
The constructor of a yacht creates two Location instances to fill the latitude and longitude members and assign them memory. I also assigned the current count of the number of Yachts to the serialNumber member and THEN incremented the counter. This is done by adding ++ to the end of counter as it will only increment by +1 after assigning the value. One issue I encountered was that I needed the counter to start from 1, and the static variable was not being initialized meaning it had no memory. To fix this I added a reset counter function which assigned the counter to number of my choosing and then assigned the value memory by setting it to NULL in the Yacht.cpp file.

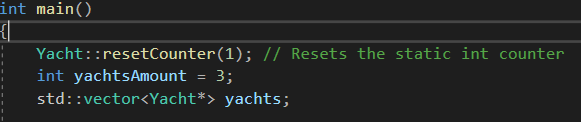
I had to make sure the counter was reset to 1 on the start of the program so I added the function into the main function now so that I don’t forget.

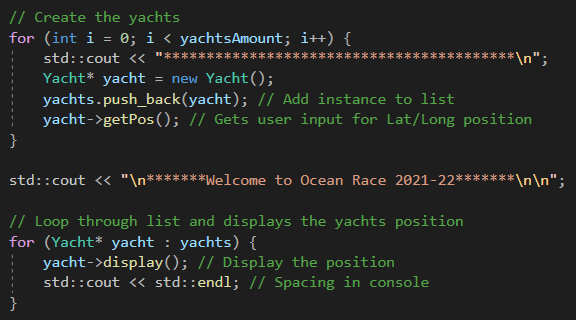


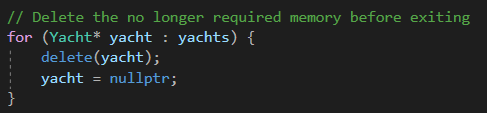
I added in the functionality for getting the latitude and longitude to make it execute the same as shown in the example in the brief.

I also did the same thing for displaying the instance of a Yacht

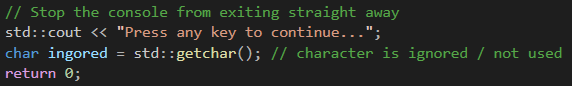
Completed Yacht class.

Now it was time to put everything together in the main function. I decided I was going to use for loops to create and execute the code each stage of the program. I also realised that we didn’t want a magic variable so I added a yacht count into the start of the program along with a vector containing pointers to Yachts to allow easy management for the Yacht throughout the programs lifetime. 

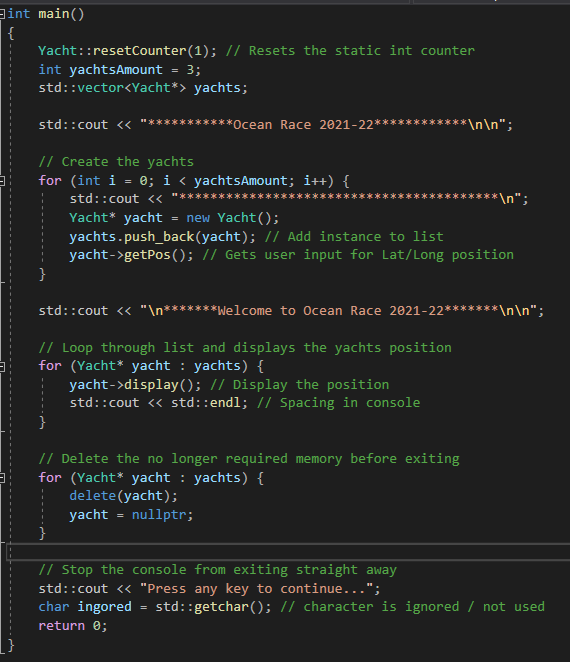
Next was the main two core loops for creating and getting a latitude and longitude, then displaying them in the race as defined in the brief.

This was made simple enough as the functions were all contained in the Yacht class and made simple work of performing these functions.

Then we had to delete all the yachts memory as we used the ‘new’ keyword:

I decided to test the program to see the results but found that the program would just exit before you could see the results when you opened the .exe directly. Debug mode allowed you to see the output but we don’t know that all persons opening the .exe will be using debug mode in Visual Studio 2019. So I decided to add on last bit of code to the end of the main function.

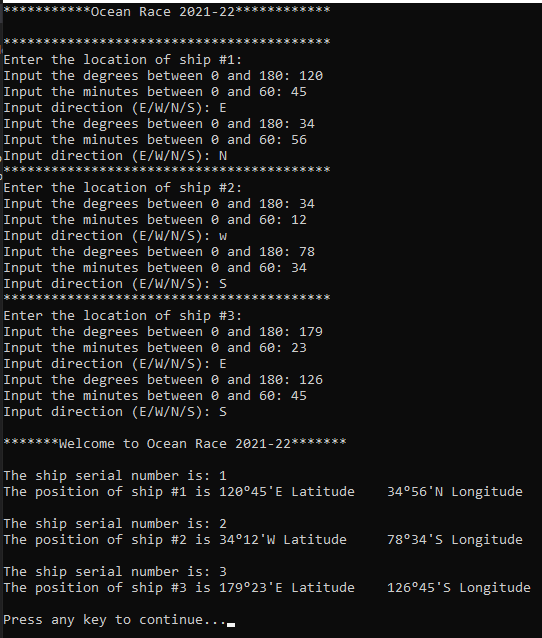
Completed main function

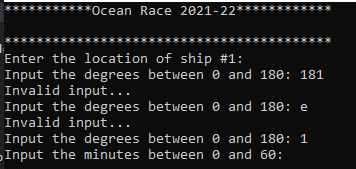


With the program put together I debugged it and ensured the program handled all types of inputs into the console correctly. The code output is shown in the next section of the document.

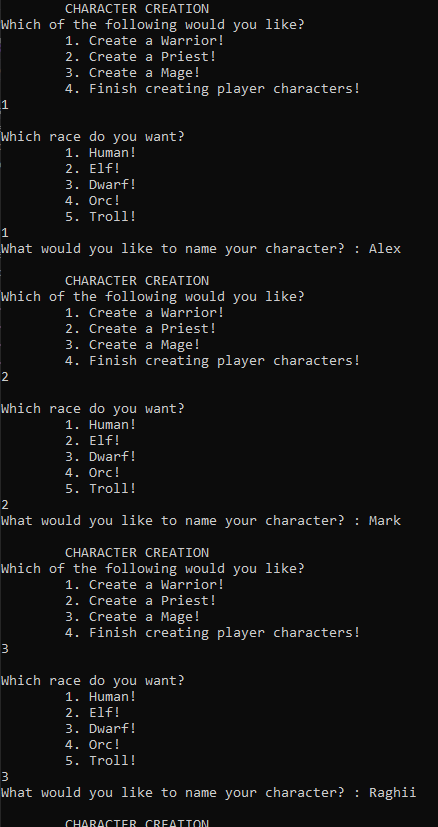
Code Screen Shots

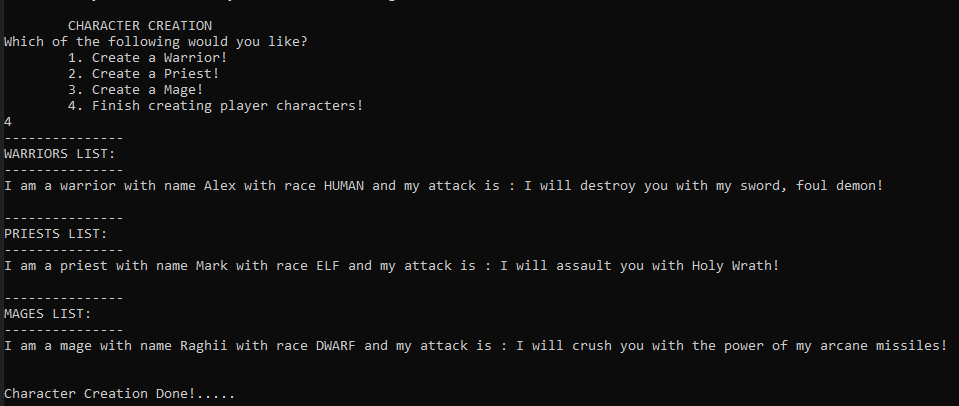
Task 1

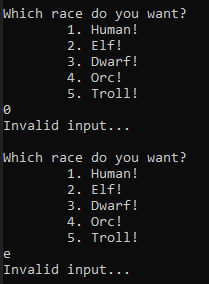
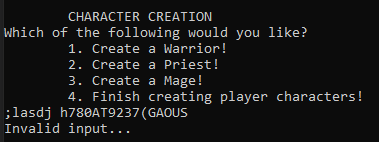




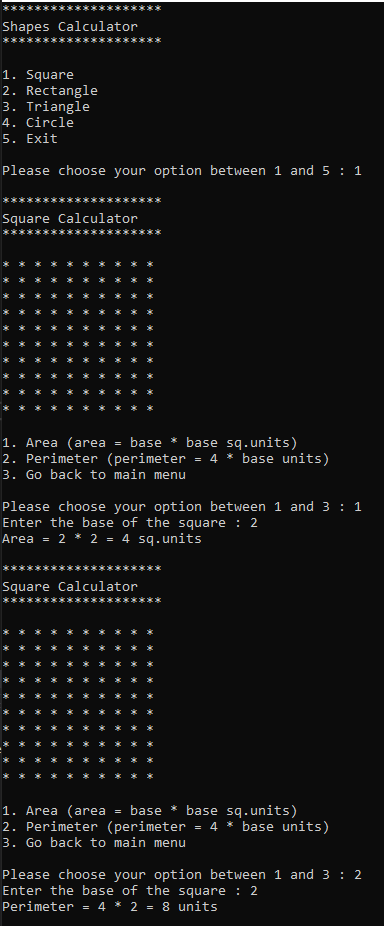
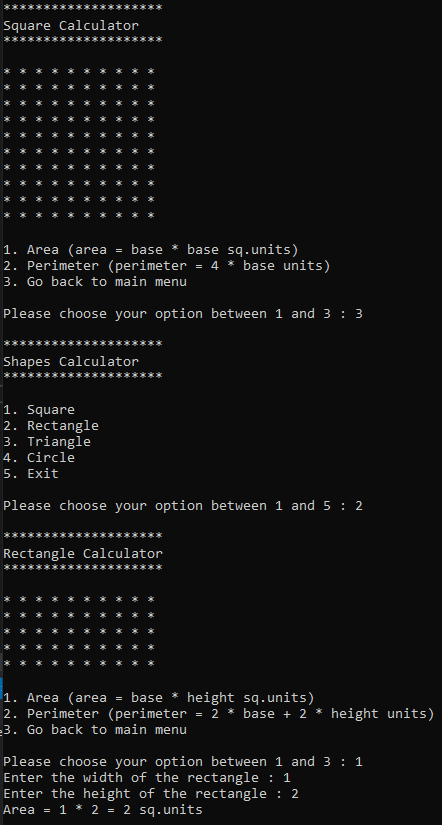
Task 2

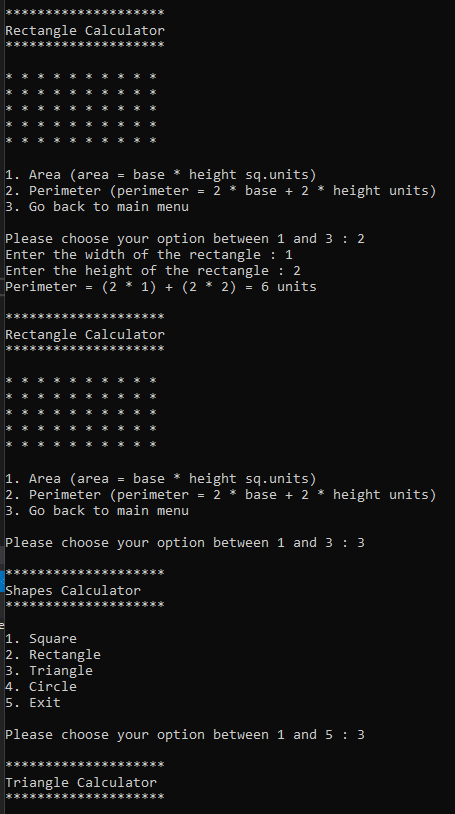


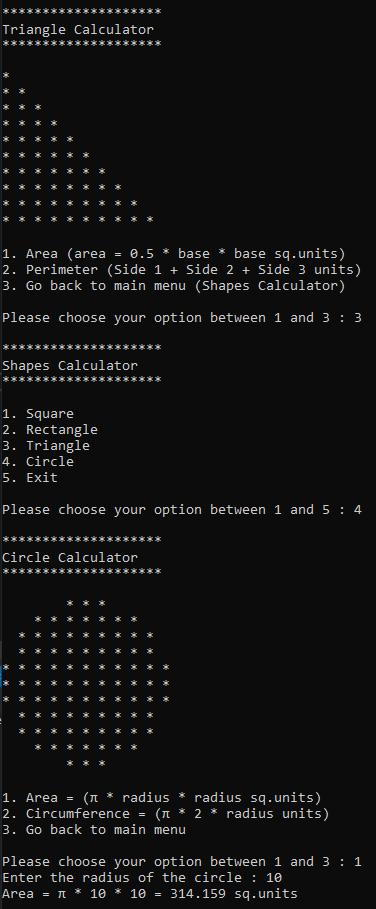
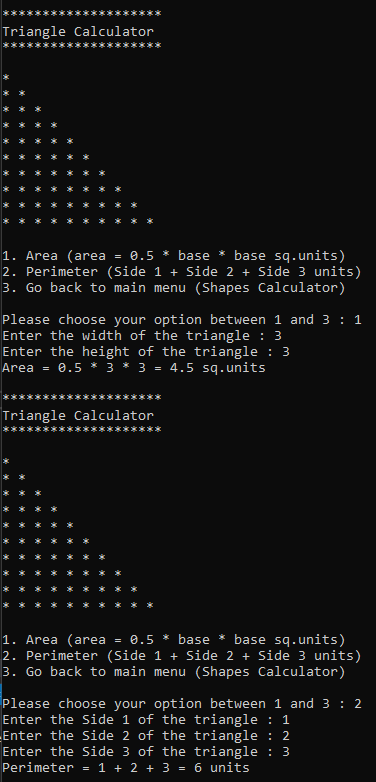


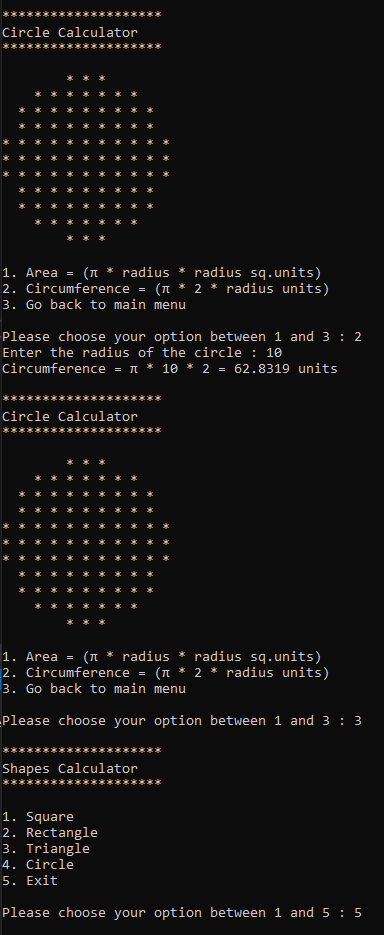
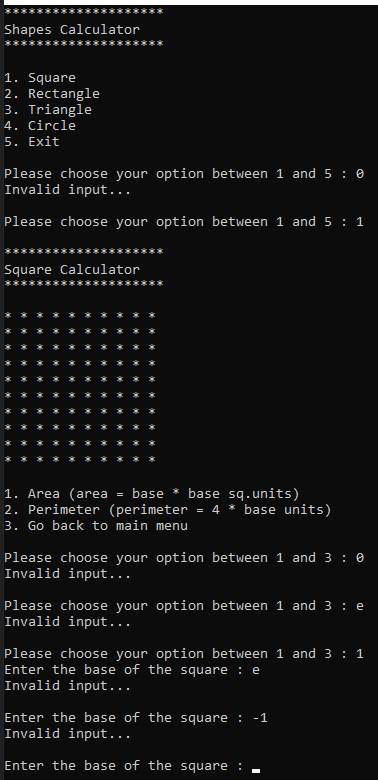
 

Task 3









Task 4

