**Student: Amir Mohamed**

**ID: 21905318**

| **Module code and title:** | CO650  Advanced Programming | **Module leader:** | Guy Walker |
| --- | --- | --- | --- |
| **Assignment No. and type:** | CW 1 | **Assessment weighting:** | 100% |
| **Submission time and date:** | 23/08/2023 | **Target feedback time and date:** | +3Wks |

GitHub Link: -

<https://github.com/Amir94Mohamed/CO650A.D.-Referal/tree/main>

Contents

[Task 1: - 3](#_Toc142704616)

[GitHubLink: - 3](#_Toc142704617)

[Project Background: - 3](#_Toc142704618)

[Class Diagram: - 3](#_Toc142704619)

[Output: - 3](#_Toc142704620)

[Abstraction 4](#_Toc142704621)

[Encapsulation 4](#_Toc142704622)

[Inheritance 5](#_Toc142704623)

[Polymorphic Behaviour 5](#_Toc142704624)

[Overloading 5](#_Toc142704625)

[Friendship 6](#_Toc142704626)

[Static Members 6](#_Toc142704627)

[Constructors 7](#_Toc142704628)

[Pointers 7](#_Toc142704629)

[Functional Pointers 7](#_Toc142704630)

[Task 1 Conclusion: - 8](#_Toc142704631)

[Extra Code Explanation : 8](#_Toc142704632)

[Task 2: - 9](#_Toc142704633)

[Classes Link: - 9](#_Toc142704634)

[Code ScreenShots 10](#_Toc142704635)

[Server.cpp 10](#_Toc142704636)

[Client.cpp 10](#_Toc142704637)

[Common.h 11](#_Toc142704638)

[Comms.h 13](#_Toc142704639)

[Console Output 16](#_Toc142704640)

[Successful connection 16](#_Toc142704641)

[Quitting Client 16](#_Toc142704642)

[Quitting server 16](#_Toc142704643)

[Client Tries to connect (server OFF) 17](#_Toc142704644)

[Server waiting for connection from Client 17](#_Toc142704645)

[Task 2 Conclusion: - 17](#_Toc142704646)

[GitHub Link: - 17](#_Toc142704647)

# Task 1: -

## GitHubLink: -

<https://github.com/Amir94Mohamed/CO650A.D.-Referal/tree/main/Task%201>

## Project Background: -

This was an old project completed during sixth form, it’s a Car Rental system that displays details about their client and what cars they have rented, its only got 1 client currently and no input ability been coded.

## Class Diagram: -

A screenshot of a computer

Description automatically generated

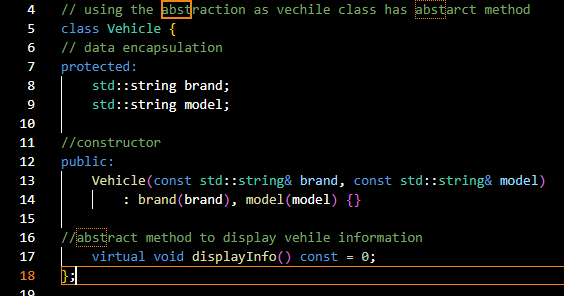
## Output: -

A screen shot of a computer

Description automatically generated

Outputs details of the client and what employee has rented the car to the client, in addition to the car model etc...

## Abstraction

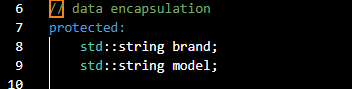


A computer screen shot of code

Description automatically generated

Abstraction is used to hide the internal implementation details, here its used to display the Car details and the customers details.

## Encapsulation

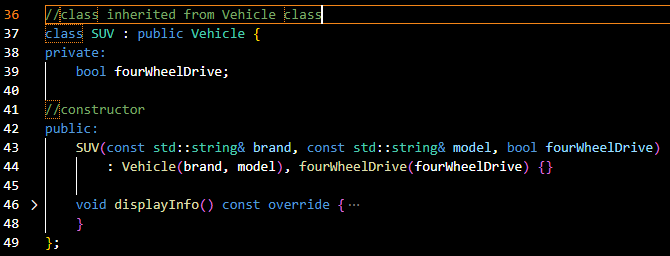


Encapsulation is used to bundle data into a location and store it for recall later, this system used it to contain the car information details and store them to access when called on.

## Inheritance

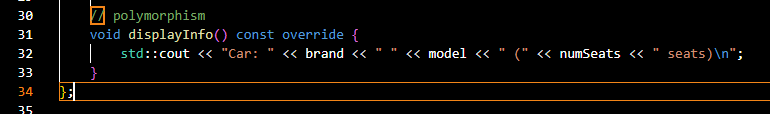
A computer screen shot of a black background

Description automatically generated



Inheritance is used to avoid making large repetitions of the code and allows the user to set data and be able to inherit its attributes when called. In this case it’s been used twice to inherit the vehicles class attributes.

## Polymorphic Behaviour



Polymorphic behaviour is used to call to a member function and cause a different function to be invoked. This case is used to display one of the 2 car rental messages.

## Overloading

A screen shot of a computer program

Description automatically generated

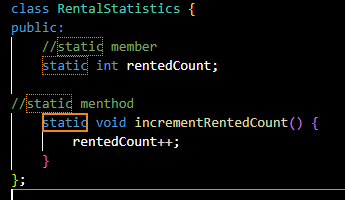
Overloading allows a method with the same name to be declared multiple times and display a different output once they have been given a different input parameter, in this case it aids in displaying the different output parameters for rented cars and how long for.

## Friendship



Friendship is used to provide access to private and public classes as a “friend”. Here it grants access to the private class of the employee’s name.

## Static Members



A Static member means no matter how many objects of the class are created, there is only one copy of the static member, this case uses the static members for the rentedCount which displays the number of rented cars for the client.

## Constructors

A computer screen shot of text

Description automatically generated

A computer code on a black background

Description automatically generated

The purpose of the constructors is to construct an object and assign values to the object’s members. In these cases, it constructs the output message and assembles the car details.

## Pointers

A computer screen shot of a program

Description automatically generated

A pointer contains a variable of an address to another variable, here it helps with the car details and ensures the correct one is displayed.

## Functional Pointers

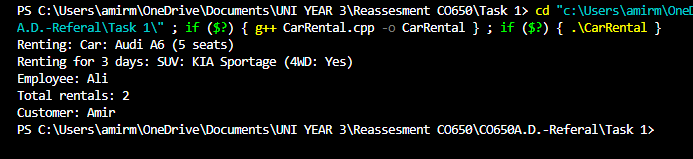
A computer screen shot of a program

Description automatically generated

Functional Pointers are used to point towards functions, this case it points towards functions that are displaying a message of the car rental status.

## Task 1 Conclusion: -

To summarize this task was a unique experience that allowed the creation of a semi functioning car rental system. Using the different points, the assignment brief provided allowed the display of a semi functioning car rental system and with a bit more work could be functional for use. Currently the program displays the rented cars (which has different way of display depending on the car), which employee provided the agreements, the total amount of rented cars and the customer’s name.



## Extra Code Explanation:

The `Vehicle` class serves as an abstract base class with a protected data section for brand and model information. It has a constructor to initialize these attributes, and it declares a pure virtual function `displayInfo()` that must be overridden by derived classes.

The `Car` class inherits from the `Vehicle` class, adding an attribute `numSeats` to represent the number of seats in the car. It overrides the `displayInfo()` method to display car information, including the number of seats.

The `SUV` class similarly inherits from `Vehicle`, but it includes an attribute `fourWheelDrive` to indicate if the SUV has four-wheel drive. It also overrides the `displayInfo()` method to display SUV information.

The `RentalManager` class demonstrates function overloading. It has two `rent` methods: one that takes a `Vehicle` parameter and one that takes both a `Vehicle` parameter and an integer representing the rental days. These methods print information about the rental.

The `Employee` class has a private attribute `name`. It is a friend of the `RentalManager` class, allowing `RentalManager` to access its private members. It includes a method `displayEmpInfo()` to display employee information.

The `RentalStatistics` class showcases the use of static members. It has a static variable `rentedCount` to keep track of the number of rentals. The static method `incrementRentedCount()` increments this count.

The `Customer` class represents a customer with a private attribute `name`. It has a method `displayCusInfo()` to display customer information.

# Task 2: -

## Classes Link: -

Server.cpp

<https://github.com/Amir94Mohamed/CO650A.D.-Referal/blob/main/Task%202%20(SERVER)/Server.cpp>

Client.cpp

<https://github.com/Amir94Mohamed/CO650A.D.-Referal/blob/main/Task%202%20(SERVER)/Client.cpp>

Comms.h

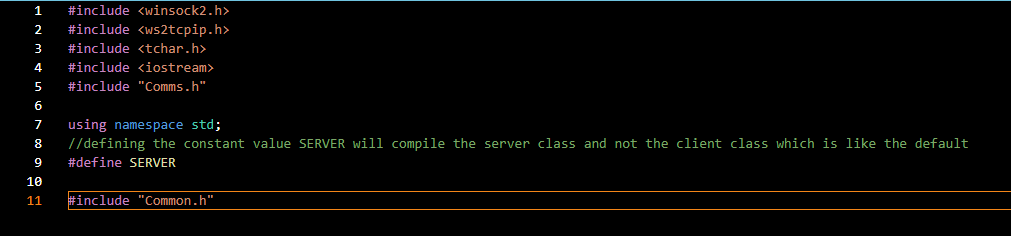
<https://github.com/Amir94Mohamed/CO650A.D.-Referal/blob/main/Task%202%20(SERVER)/Comms.h>

Common.h

<https://github.com/Amir94Mohamed/CO650A.D.-Referal/blob/main/Task%202%20(SERVER)/Common.h>

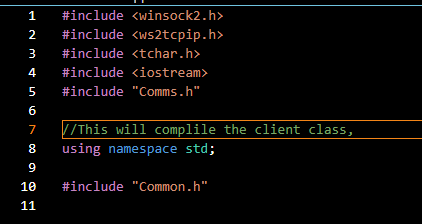
## Code Screenshots

### Server.cpp



This is used to open a server that a client can connect too, it uses the Comms.h & Common.h files to do the connections.

### Client.cpp



This code is what ensures the client code is running which is the default. This is used to open a client chat that can connect to the server and type, it uses the Comms.h & Common.h files to do the connections.

### Common.h

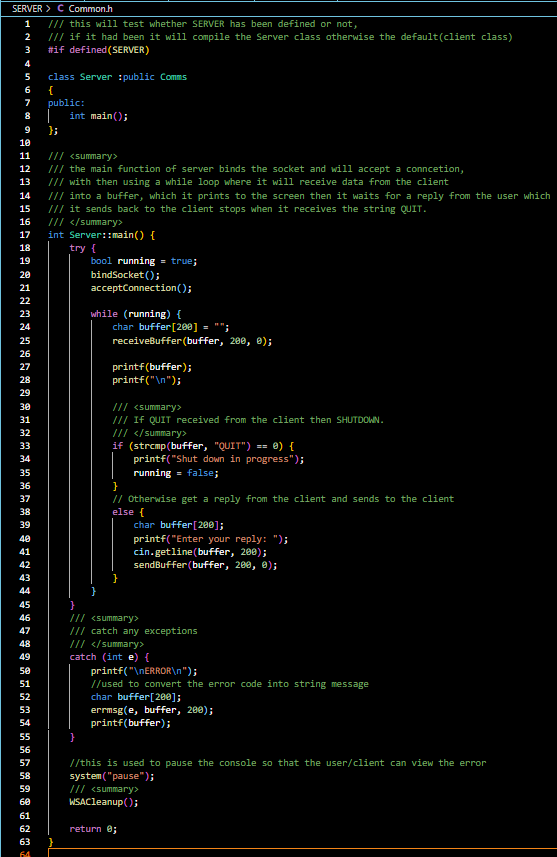
Common.h aids with the connections established between the client and server. It contains two classes, Server, and Client, which are derived from the Comms class. The code defines the main functions of both classes.

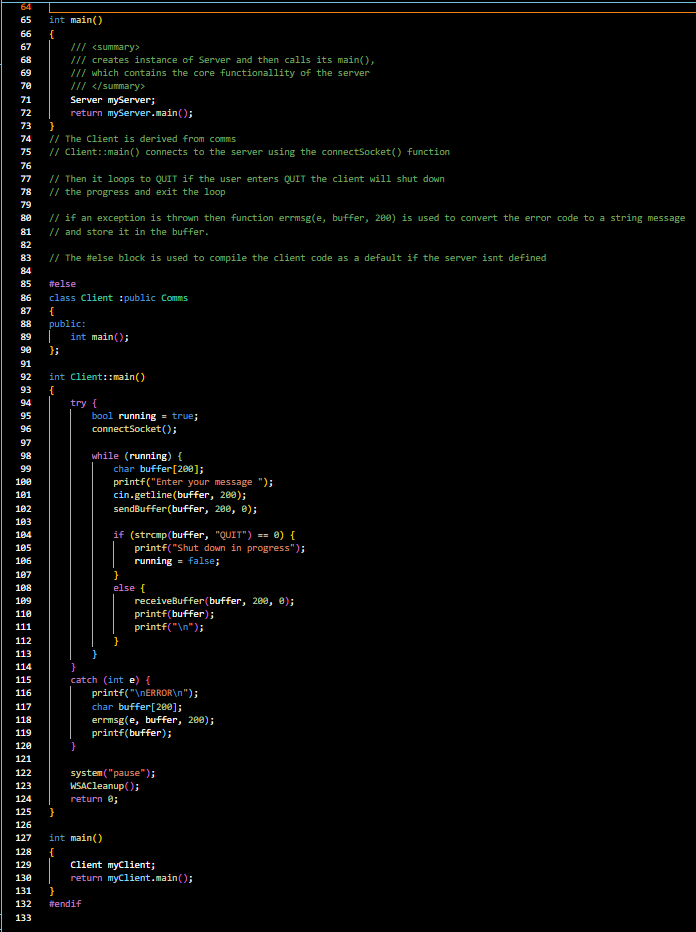
Server Class: -

* Binds the socket and waits for a connection from a client.
* Enter a while loop to receive data from the client into the buffer.
* Prints received data to the screen.
* Checks if the received data is “QUIT”. If yes it shuts down.
* If data isn’t “QUIT” it prompts the user to enter a reply and sends it back to the client
* If any exception occurs, it catches the exception, converts the error code into a string message and prints it.
* The main function of the server class returns 0 after the while loop exits.
* The main function of the program creates an instance of the Server class and calls its main function.

Client Class: -

* Connects to the server using the connectSocket() function.
* Enters a while loop to send messages to the server.
* Prompts the user to enter am message and sends it to the server.
* Checks if the sent message is “QUIT”. If it is, the clients shuts down.
* If the sent message is not “QUIT”, it receives a reply from the server and prints it.
* If any exceptions occurs, it catches the exception, converts the error code into a string message and prints it.
* The main function of the client class returns 0 after the while loop exits.
* The main function of the program creates an instance of the client class and calls its main function.





### Comms.h

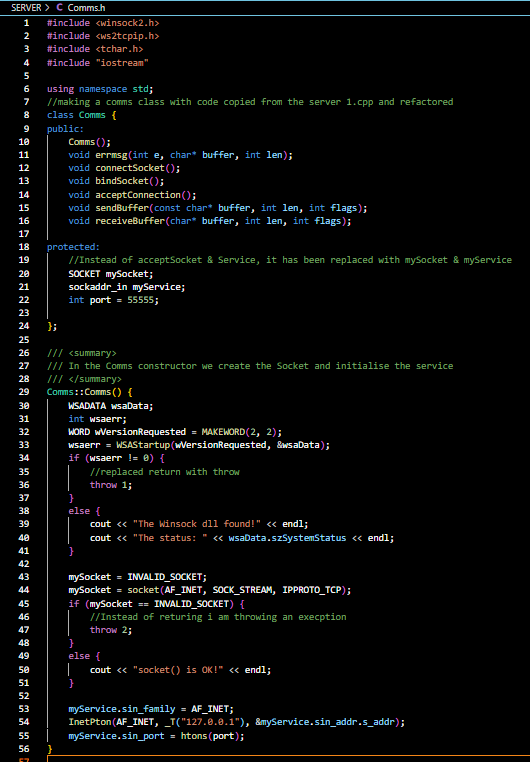
The comms.h file oversaw the communications between the client and server. Specifically, the communications that need to be established for the connection to succeed such as creating the sockets, accepting and error message if failed connection. These functions in the class Comms create the WS2(Windows Sockets version 2) creating a socket, connecting to a server (connectSocket), binding a socket to a port on the host PC (bindSocket), accepting connections from other clients (acceptConnection), sending data (sendBuffer), and receiving data (receiveBuffer).

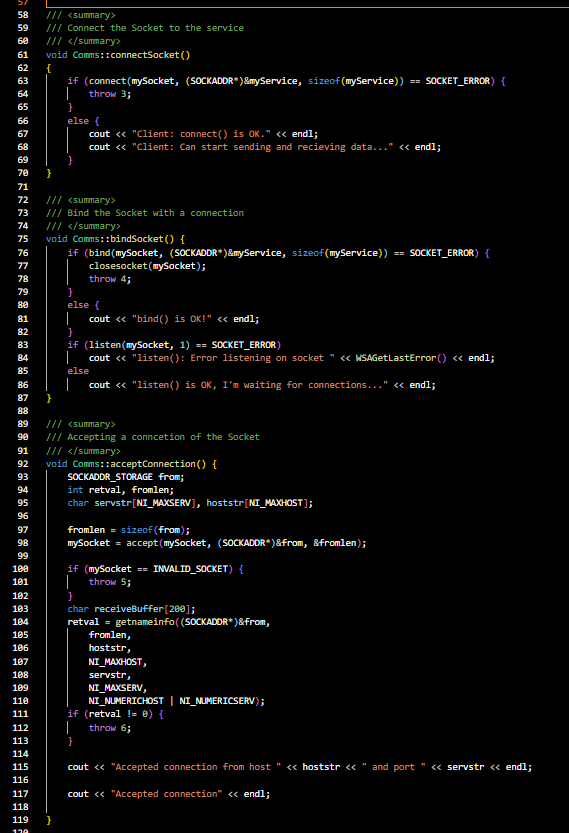
Regarding connectSocket(): - It is used to connect to remote server using connect().

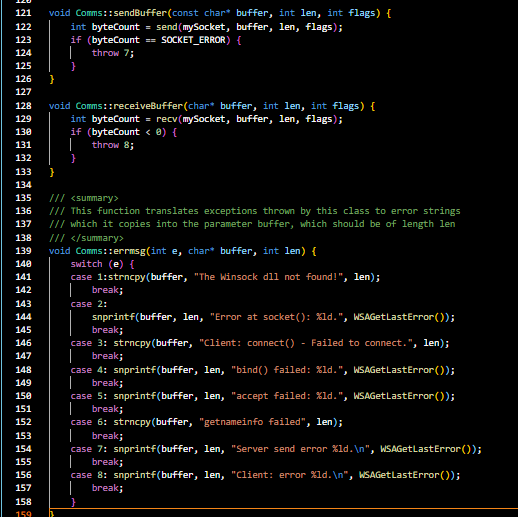
Regarding bindSocket(): - It binds the socket to an IP address and port using bind(), then starts listening for connections using listen().

Regarding acceptConnection(): - It accepts connections from other sockets.

Regarding sendBuffer(): - It sends data over the socket. Regarding receiveBuffer(): - It receives data from the socket.

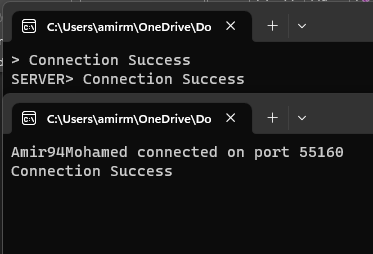




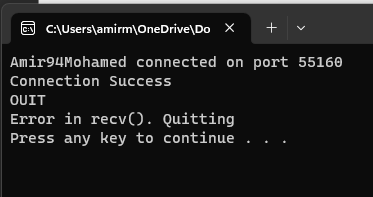


## Console Output

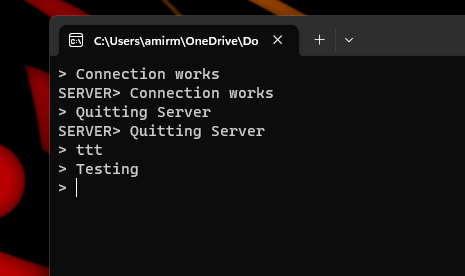
### Successful connection



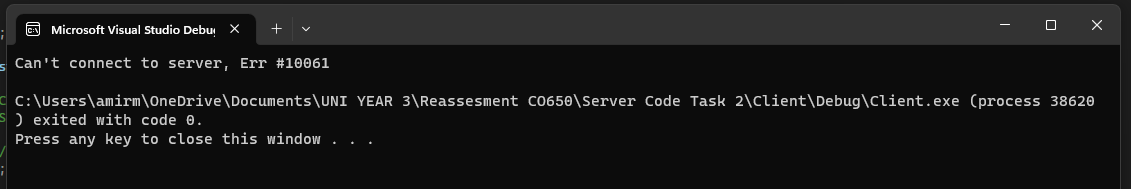
### Quitting Client



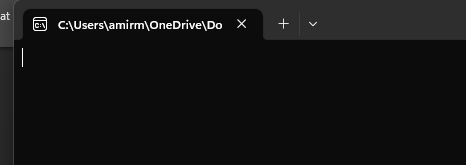
### Quitting server



### Client Tries to connect (server OFF)



### Server waiting for connection from Client



## Task 2 Conclusion: -

To summarize this was a good experience and practice, It was quite unique and required further research into the task to complete it. The main issue was the C++ language and the lack of experience in the language. The result was functional but had some issues that would cause it to display the wrong message but eventually fixed by displaying a different message. To aid in the development of this task, A website by Bhalla, W. (2021) “What is Winsock and how does it work?” was able to provide a decent understanding of what and how Winsock works. In addition to many YouTube videos that aided in the understanding of C++ Coding.

## Task 2 Reference: -

Bhalla, W. (2021a) *What is Winsock and how does it work?*, *MUO*. Available at: <https://www.makeuseof.com/what-is-winsock/> (Accessed: 17 August 2023).

## GitHub Link: -

<https://github.com/Amir94Mohamed/CO650A.D.-Referal/tree/main/Task%202%20(SERVER)>