YOOBEE
COLLEGE OF CREATIVE INNOVATION

Diploma in Software Development

CS105

Development Principles II

(NZQF Level 5, 15 credits)

Assessment 1

Case Study Report

Weighting within course 50%

Candidate's Assessment Instructions

- This assessment is an ***open book activity***, you can use your own course and review notes as well as offline or online resources, such as textbooks or online journals.
- You will each submit the identical piece of work separately for the work that requires collaboration.
- You can always ask your online tutor if you need further explanation or if the instructions are not clear.
- The purpose of this assessment is to assess your knowledge. As part of your academic and professional integrity, you must work alone on this assessment. In the event Yoobee suspects collusion, this will be addressed. For more information on plagiarism, please refer to the Student Handbook.
- Submit your completed assessment online in the correct space provided.
- Marks and feedback will be returned within 15 days of the submission date.

Submission/Deliverables of your Work

Submissions are accepted only via a Yoobee Online and all files should be submitted in a single zip file including:

- A word document which consists of code, output screenshots, UML diagrams and any other explanations/content asked in the assessment tasks. The word document must have your name, student id on the headers. Page numbers must be included in the footers. Use a font of either Times New Roman or Calibri. The minimum font-size must be 12 with at least 1.15 line spacing. Please ensure all diagrams, tasks, figures, or tables (if any) are numbered.
- A file containing the C++ solutions of all programming tasks asked in this assessment. Every C++ file must have comments with question number and full name.

Success Criteria

You need to achieve **50% or more** of the marks to pass this course.

● You are allowed a maximum of three attempts. The maximum percentage to be awarded on a second and third assessment attempts is 50%.

Learning outcomes (LO):

1. Explore key principles, methods, tools and frameworks used in the development of mobile games.
2. Describe the processes used to solve development problems.
3. Demonstrate developing literacy in key programming languages for mobile game development.
4. Review individual learning, practices, and strategies as developers of mobile games

Scenario 1: (LO1, LO2, LO3)

The Ocean Race 2021-22 route is announced. Yachts are ready to sail and will visit 10 international cities, including the start port and the Grand Finale finish in Genoa, Italy. In ocean navigation, degrees and minutes (for latitude and longitude) are used to measure locations along with the direction. Degrees of latitude are either north (N) or south (S) measured from 0 to 180 degrees and degrees longitude are either east (E) or west (W) measured from 0 to 90.

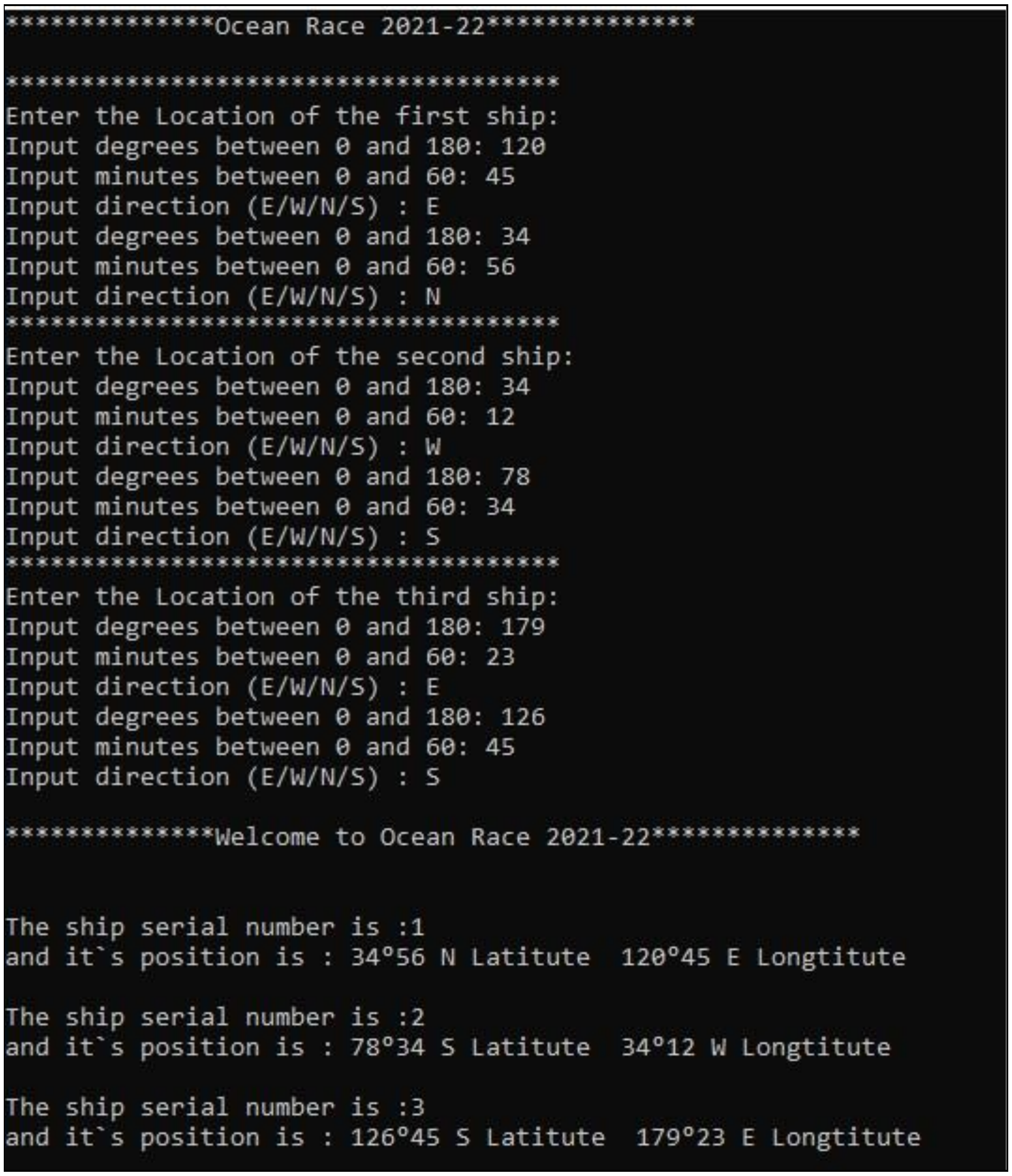
Example: 38°56'N, 71°0'W is 38 degrees 56 minutes North and 71 degrees 0 minutes West.

Task:

1. Your task is to create a ***class Yacht*** that includes the yacht's *number* and *location* ***(LO3).***
    a) For Yacht's Number : You have to number each ***yacht*** *object* as it is created using a ***constructor***. To do this, you will have a data member that holds a serial number for each object created from the class. Then you will need another data member that records a count of how many objects have been created so far.
    b) For Yachts Location : Create a *class* ***Location*** that has three member variables --
        i) int for *degrees* (explained above)
        ii) float for *minutes* (explained above)
        iii) Char for d*irection* letters N,S,E,W

        Location class has variables like *longitude* and *latitude*. *Location class* includes one member ***getpos()*** which obtains a location value in degrees (between 0-180) and minutes (between 0 to 60) and direction (N,S,E,W) from the user.
    c) One member function [***get_pos()*** *]* of the ***Yacht*** *class* should get a position from the user and store it in the object; Another member function [***display()*** *]* should report the yacht number and location. ***display()*** use two variables from ***Location class*** to represent the yacht's location, *latitude* and *longitude*. ***display()*** function will display the location latitude and longitude in *148°26' N format.*
    d) The ***main()*** program creates three yachts, asks the user to input the location and then displays each yacht's number and location.
2. Prepare the UML class diagram for the *Location class* ***(LO1).***
3. Clearly describe the process used to solve the problem with the help of appropriate comments that increases the readability of the program **(LO2**).

Note: Use the hex character constant '\xF8' to display a degree symbol (°). **Sample output :**



```
**************Ocean Race 2021-22**************

*************************************
Enter the Location of the first ship:
Input degrees between 0 and 180: 120
Input minutes between 0 and 60: 45
Input direction (E/W/N/S) : E
Input degrees between 0 and 180: 34
Input minutes between 0 and 60: 56
Input direction (E/W/N/S) : N
*************************************
Enter the Location of the second ship:
Input degrees between 0 and 180: 34
Input minutes between 0 and 60: 12
Input direction (E/W/N/S) : W
Input degrees between 0 and 180: 78
Input minutes between 0 and 60: 34
Input direction (E/W/N/S) : S
*************************************
Enter the Location of the third ship:
Input degrees between 0 and 180: 179
Input minutes between 0 and 60: 23
Input direction (E/W/N/S) : E
Input degrees between 0 and 180: 126
Input minutes between 0 and 60: 45
Input direction (E/W/N/S) : S

**************Welcome to Ocean Race 2021-22**************


The ship serial number is :1
and it`s position is : 34°56 N Latitute  120°45 E Longtitute

The ship serial number is :2
and it`s position is : 78°34 S Latitute  34°12 W Longtitute

The ship serial number is :3
and it`s position is : 126°45 S Latitute  179°23 E Longtitute
```

Scenario 2: (LO 1, LO 3)

Suppose that you are part of a team which is creating a role-playing game (RPG). You have been asked to create the foundational classes to represent player characters, that is, the entity within the game that the user of the game software controls.

The Player Base Class: LO3

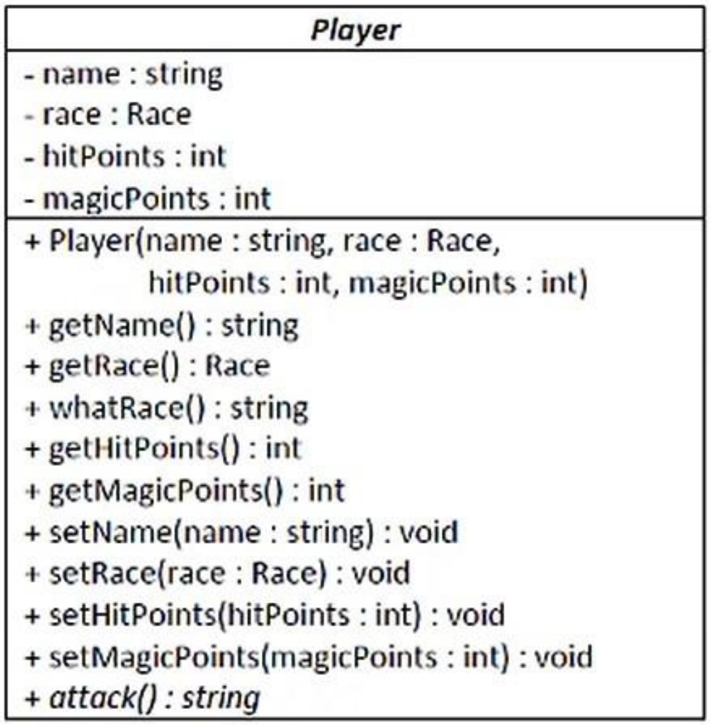
- The Player base class contains name, race, hitPoints, and magicPoints data members for the players.
- It provides a constructor with four parameters corresponding to all of the four data members.
- It provides getters and setters member functions for all the four data members.
- These four data members will be common to all Player objects.
- The Player class will contain an attack() member function that returns "No attack method defined yet!".
  Note: The derived classes will override this attack() member function to define specific attack methods for Warrior, Priest, and Mage derived classes as given in next sub-section.

The enum type: *LO 1*

- The race can be represented by an enumerated type using enum with the following values:
  - HUMAN
  - ELF
  - DWARF
  - ORC
  - TROLL

Note: The enum may be code inside Player.h which will ensure all derived classes have access to it.

Below is a UML class diagram only for the player base class.



Player

- name : string
- race : Race
- hitPoints : int
- magicPoints : int

+ Player(name : string, race : Race, hitPoints : int, magicPoints : int)
+ getName() : string
+ getRace() : Race
+ whatRace() : string
+ getHitPoints() : int
+ getMagicPoints() : int
+ setName(name : string) : void
+ setRace(race : Race) : void
+ setHitPoints(hitPoints : int) : void
+ setMagicPoints(magicPoints : int) : void
+ *attack() : string*

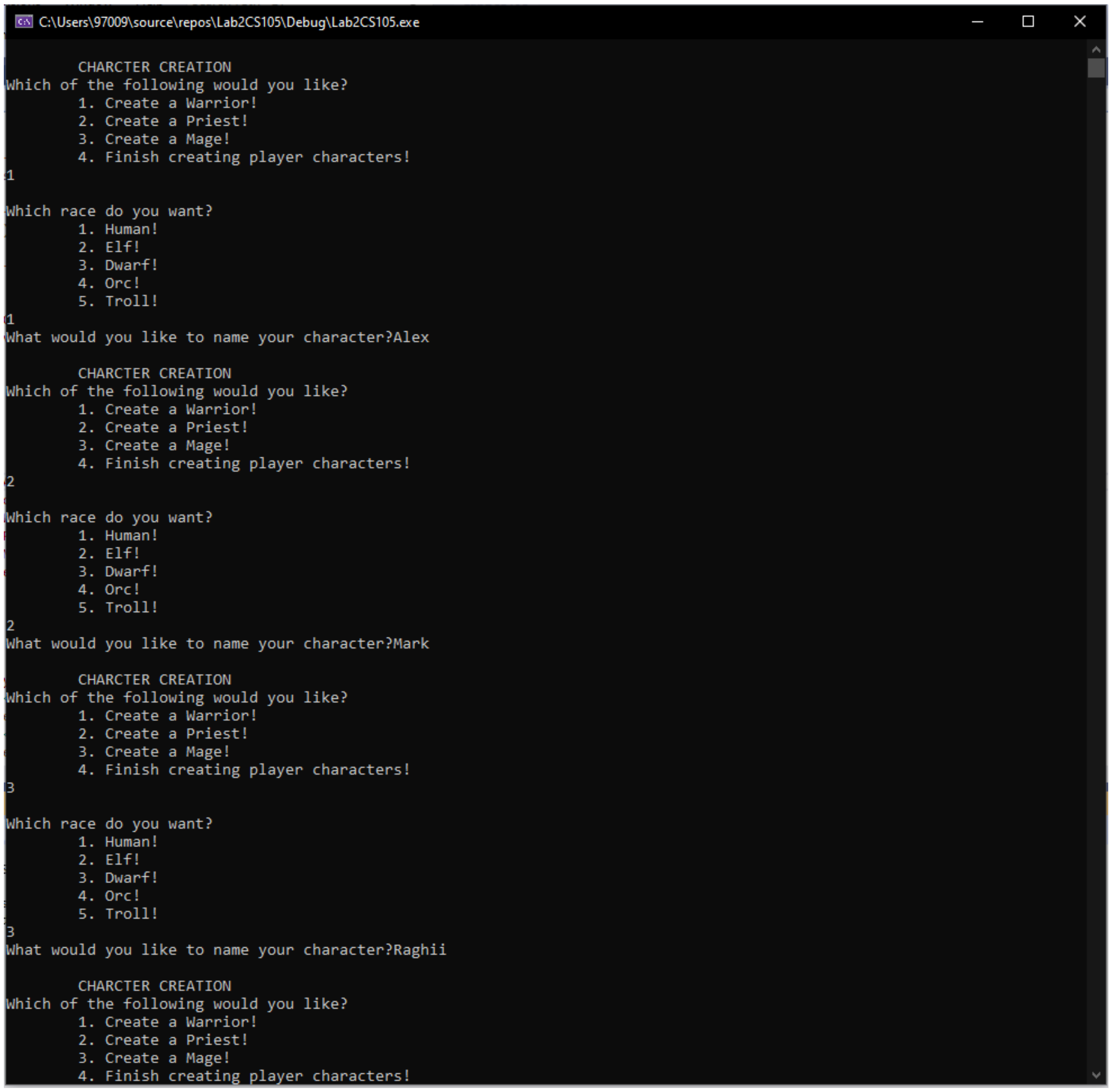
The Three Derived Classes:
Following are the specific player types that will be derived classes of the Player class:

- **Warrior**
  - 200 hitPoints, 0 magicPoints
  - Their attack method should return “I will destroy you with my sword!”
- **Priest**
  - 100 hitPoints, 200 magicPoints
  - Their attack method should return “I will assault you with holy wrath!”
- **Mage**
  - 200 hitPoints, 0 magicPoints
  - Their attack method should return “I will crush you with my arcane missiles!”

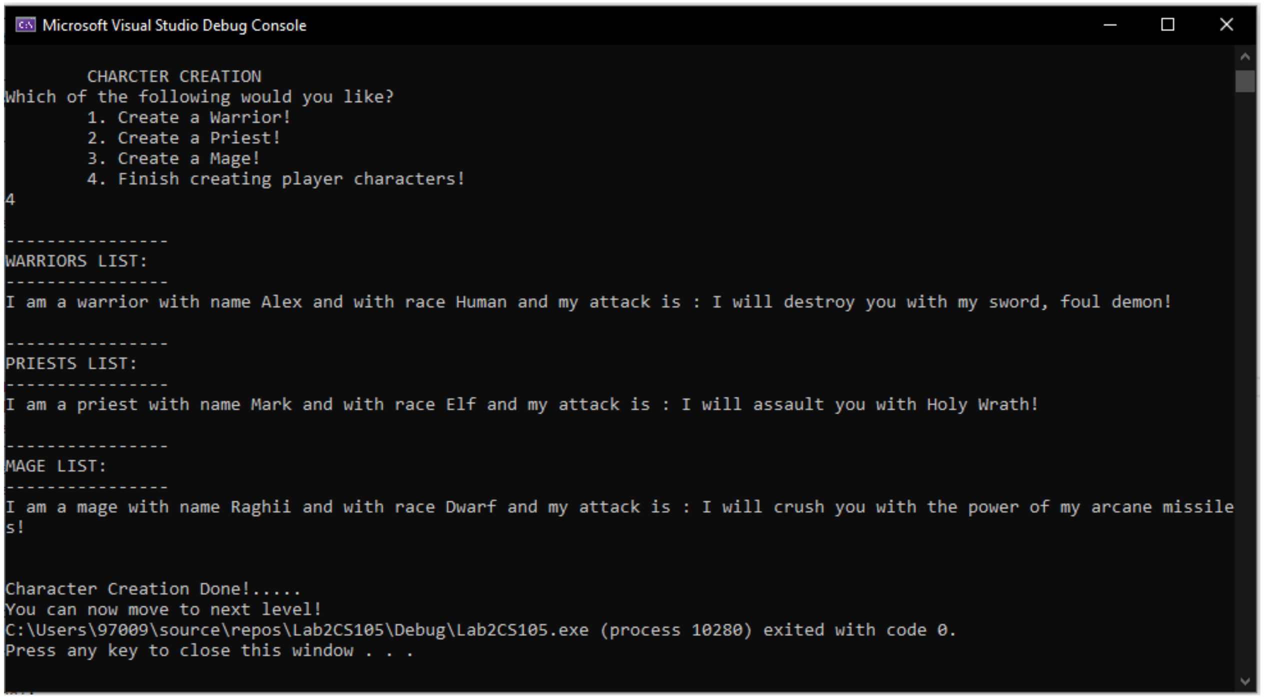
The main function:
Inside the main function, you should allow the user to create different kinds of player objects, allowing them to select the derived class which you can request from the user using a simple menu. You should also let the user select the race for their player as well. Additionally, you can ask the user to assign a name to the player as well.

Store players are created by the user in three different vectors namely warrior, priest, and mage. Once you are done with the creation of player objects, iterate through all three vectors and print out the player details along with their attack methods, names, and race ***(LO 4)***.

Sample Output:



```
C:\Users\97009\source\repos\Lab2CS105\Debug\Lab2CS105.exe

        CHARCTER CREATION
Which of the following would you like?
        1. Create a Warrior!
        2. Create a Priest!
        3. Create a Mage!
        4. Finish creating player characters!
1

Which race do you want?
        1. Human!
        2. Elf!
        3. Dwarf!
        4. Orc!
        5. Troll!
1
What would you like to name your character?Alex

        CHARCTER CREATION
Which of the following would you like?
        1. Create a Warrior!
        2. Create a Priest!
        3. Create a Mage!
        4. Finish creating player characters!
2

Which race do you want?
        1. Human!
        2. Elf!
        3. Dwarf!
        4. Orc!
        5. Troll!
2
What would you like to name your character?Mark

        CHARCTER CREATION
Which of the following would you like?
        1. Create a Warrior!
        2. Create a Priest!
        3. Create a Mage!
        4. Finish creating player characters!
3

Which race do you want?
        1. Human!
        2. Elf!
        3. Dwarf!
        4. Orc!
        5. Troll!
3
What would you like to name your character?Raghii

        CHARCTER CREATION
Which of the following would you like?
        1. Create a Warrior!
        2. Create a Priest!
        3. Create a Mage!
        4. Finish creating player characters!
```



Microsoft Visual Studio Debug Console

```
        CHARCTER CREATION
Which of the following would you like?
        1. Create a Warrior!
        2. Create a Priest!
        3. Create a Mage!
        4. Finish creating player characters!
4

----------------
WARRIORS LIST:
----------------
I am a warrior with name Alex and with race Human and my attack is : I will destroy you with my sword, foul demon!

----------------
PRIESTS LIST:
----------------
I am a priest with name Mark and with race Elf and my attack is : I will assault you with Holy Wrath!

----------------
MAGE LIST:
----------------
I am a mage with name Raghii and with race Dwarf and my attack is : I will crush you with the power of my arcane missile
s!


Character Creation Done!.....
You can now move to next level!
C:\Users\97009\source\repos\Lab2CS105\Debug\Lab2CS105.exe (process 10280) exited with code 0.
Press any key to close this window . . .
```

Scenario 3: (L01, L02, L04)

Bob Jones is a building architect hired to build a mall in the city centre of Chicago. He needs to produce the building plan to his team. He wants to calculate the area and perimeter of some of the structures he is planning to build which is part of the software he uses. Details are as follows:

a) A square room
b) A rectangular room
c) A circular swimming pool
d) A triangular portico

He wants the app to allow him to enter different dataset to see if it can all fit well in the total available land area.

Task:
You are to draw the class diagram UML clearly depicting the access specifiers, inheritance, data members and member functions of all classes planned to use. Then write a program in C++ to incorporate the following. ***(L01, LO4)***

Implementation details:

- Parent class - Shapes
- Child classes – Square, Rectangle, Triangle, circle
- Use functions to draw the lines, shapes of each selected item as shown in sample output to facilitate the reuse of the code.
- Create appropriate menus such as Shapes Calculator, Square Calculator, Rectangle Calculator etc.
- Choose appropriate access specifiers for data members and data functions to

arrive at the output shown in the sample.

- Similarly getData(), calculateArea() and giveResult() member functions are useful in the parent class.
- Again calculatePerimeter() in children classes are more appropriate as they can inherit the above mentioned 3 functions from the parent.

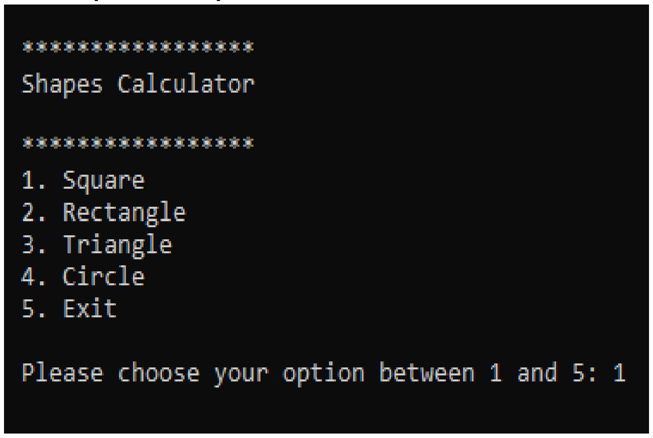
Typical members of the classes:

Three data members of double data type namely base, height and result may be helpful in the parent class.

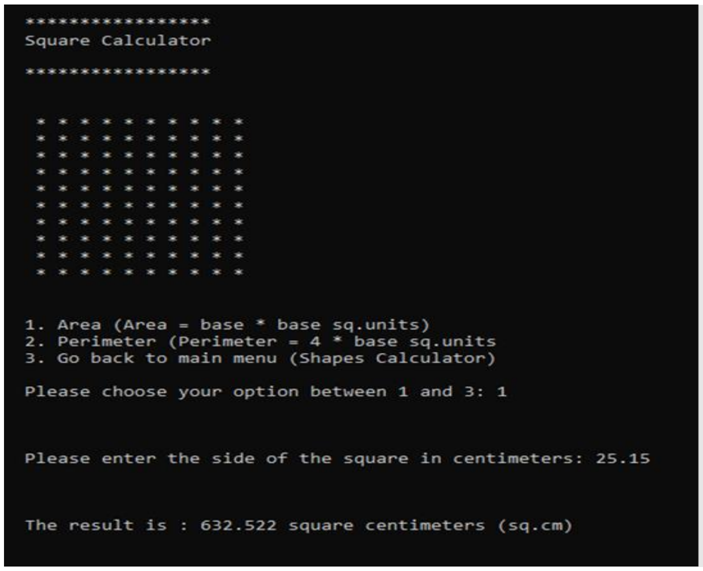
Note: **You are free to choose your data members to demonstrate the inheritance and polymorphism specifically function overloading and function overriding.**

Write comments to show the parent class, child classes, access specifiers, constructors, data members, member functions, inheritance, and polymorphism- function overloading and function overriding ***(LO 2).***

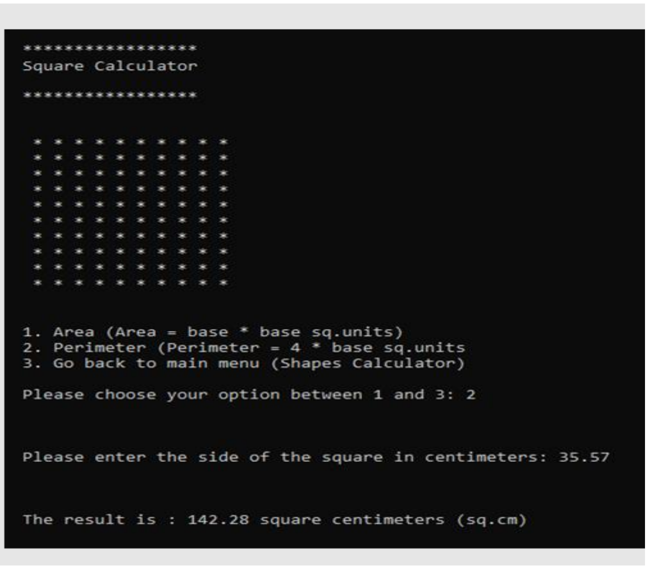
Sample output



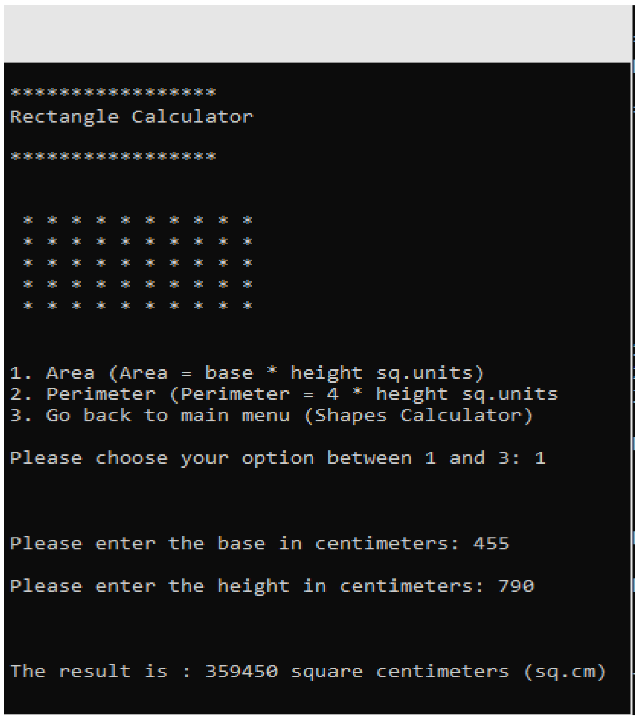
```
*****************
Shapes Calculator

*****************
1. Square
2. Rectangle
3. Triangle
4. Circle
5. Exit

Please choose your option between 1 and 5: 1
```



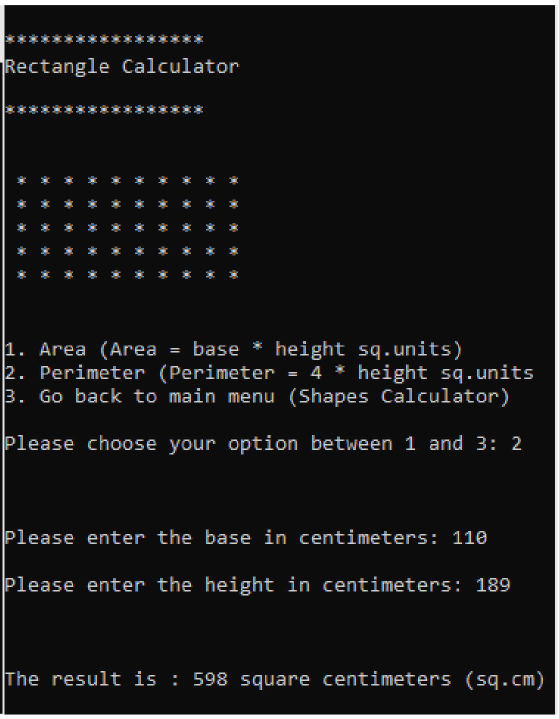
```
*****************
Square Calculator

*****************


 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *


1. Area (Area = base * base sq.units)
2. Perimeter (Perimeter = 4 * base sq.units
3. Go back to main menu (Shapes Calculator)

Please choose your option between 1 and 3: 1


Please enter the side of the square in centimeters: 25.15


The result is : 632.522 square centimeters (sq.cm)
```



```
*****************
Square Calculator

*****************


 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *


1. Area (Area = base * base sq.units)
2. Perimeter (Perimeter = 4 * base sq.units
3. Go back to main menu (Shapes Calculator)

Please choose your option between 1 and 3: 2


Please enter the side of the square in centimeters: 35.57


The result is : 142.28 square centimeters (sq.cm)
```



```
*****************
Rectangle Calculator

*****************


 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *


1. Area (Area = base * height sq.units)
2. Perimeter (Perimeter = 4 * height sq.units
3. Go back to main menu (Shapes Calculator)

Please choose your option between 1 and 3: 1


Please enter the base in centimeters: 455

Please enter the height in centimeters: 790


The result is : 359450 square centimeters (sq.cm)
```



```
*****************
Rectangle Calculator

*****************


 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *
 * * * * * * * * * *


1. Area (Area = base * height sq.units)
2. Perimeter (Perimeter = 4 * height sq.units
3. Go back to main menu (Shapes Calculator)

Please choose your option between 1 and 3: 2


Please enter the base in centimeters: 110

Please enter the height in centimeters: 189


The result is : 598 square centimeters (sq.cm)
```

Scenario 4: (L 01, L 03)

Suppose that you are part of a team which is creating a game universe for an Alien based game. ***L 03***

The Alien class in the game will store height, weight, and gender of some existing Aliens and will use operator overloading to perform breeding among Aliens using the overloading of following operator:

- + operator will mean "breeding" keeping the following rules in mind:
  - *Weight of offspring = (sum of parents' weight)/2*
  - *Height of offspring = (sum of parents' heights)/2*
  - *Gender = Use 50% chance for female and 50% chance for male*
  - *(rand() function can be used)*

- Additionally, the Alien class will perform the overloading of following operators for comparing prestige between any two given Aliens.
  - == and !=
  - and >=
  - < and <=

- Also, the following operator can be overloaded for assignment of one Alien object to another Alien object.
  - =

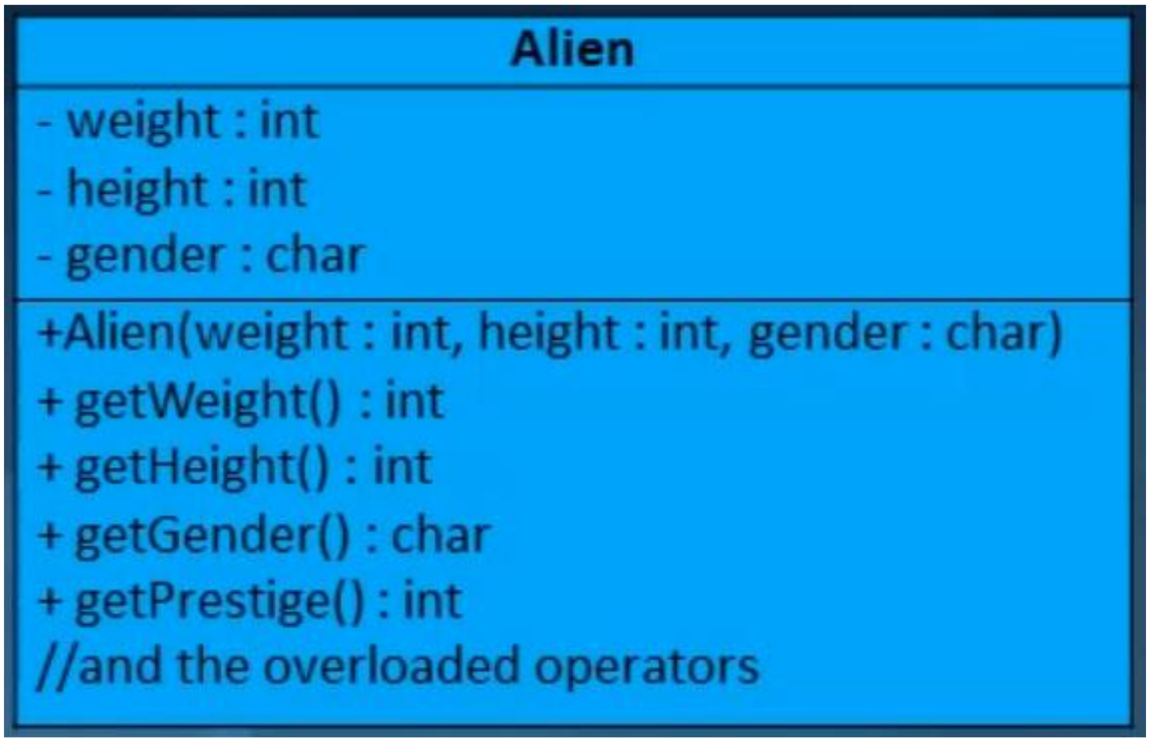
The Alien class will have the following data members and member functions:

The Alien Class:

- The Alien class contains weight, height, and gender.
- It provides a constructor with three parameters corresponding to all the three data members.
- It provides getter member functions for all the three data members.

- Additionally, there is another member function called getPrestige() which uses the following formula to calculate a prestige value for every alien.
  - *Prestige is calculated as: p = height * weight * genderPoints*
    - *where genderPoints for male = 2 and for female = 3.*

The class diagram is given below.



The main function: *L 01*

Inside the main function, you should allow the user to create Alien 1 (a male), Alien 2 (a female), Alien 3 (a male), Alien 4 (a female).

Initialization can be used for Aliens or a menu can be created for Alien creation.

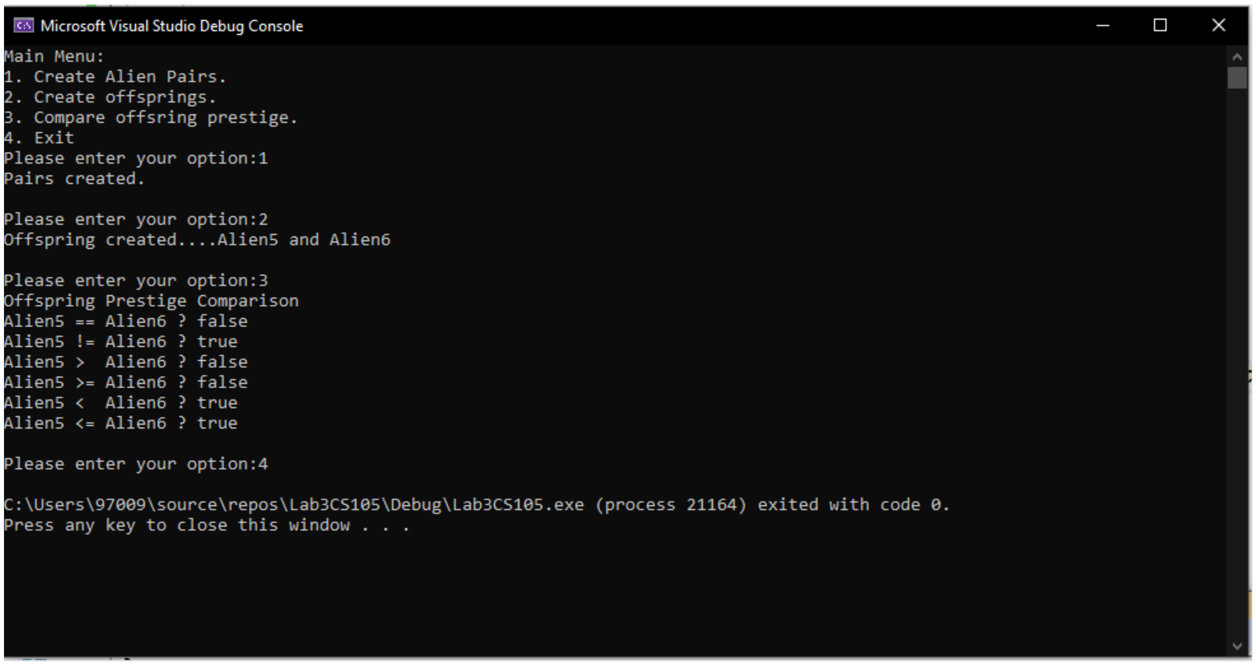
Additionally, you should create offspring called "Alien 5" from Alien 1 and Alien 2, and another offspring called "Alien 6" from Alien 3 and Alien 4.

Once all the alien characters are created, you should compare the "prestige" of offspring Alien 5 and Alien 6 using the following overloaded operators:

- Alien 5 == Alien 6

- Alien 5 != Alien 6
- Alien 5 > Alien 6
- Alien 5 < Alien 6
- Alien 5 >= Alien 6
- Alien 5 <= Alien 6

Sample Output:



```
Main Menu:
1. Create Alien Pairs.
2. Create offsprings.
3. Compare offsring prestige.
4. Exit
Please enter your option:1
Pairs created.

Please enter your option:2
Offspring created....Alien5 and Alien6

Please enter your option:3
Offspring Prestige Comparison
Alien5 == Alien6 ? false
Alien5 != Alien6 ? true
Alien5 >  Alien6 ? false
Alien5 >= Alien6 ? false
Alien5 <  Alien6 ? true
Alien5 <= Alien6 ? true

Please enter your option:4

C:\Users\97009\source\repos\Lab3CS105\Debug\Lab3CS105.exe (process 21164) exited with code 0.
Press any key to close this window . . .
```

Performance Criteria (Rubric)

Assessment Criteria & Weighting		D Range	C Range	B Range	A Range
			Scenario 1 – Yatch Game (LO1, LO2, LO3) 14%		
1a) Yatch's constructor	**2%**	The constructor's implementation is incorrect or has not been implemented at all. Comments are missing or have ambiguity.	The constructor has been correctly implemented but variable naming conventions are inconsistent and not meaningful. Comments are missing or have less clarity.	The constructor has been correctly implemented along with meaningful and consistent variable naming conventions. Comments are present but have less clarity.	The constructor has been correctly implemented along with meaningful and consistent variable naming conventions. Comments that provide clarity to the code are present.
1b) Location class instance variables	**2%**	The class has not been correctly implemented or has not been implemented by the student.	The class has been correctly implemented with the required instance variables. The member method getpos() has been correctly defined with the correct return type and no comments that explain the method's purpose are present.	The class has been correctly implemented with the required instance variables. The member method getpos() has been correctly defined with the correct return type and comments that explain the method's purpose are present with less clarity.	The class has been correctly implemented with the required instance variables. The member method getpos() has been correctly defined with the correct return type and comments that provide clarity to the method's purpose.
1c) Location class Member functions (method)	**4%**	All member functions (methods) of the class are either not implemented or partially implemented. Comments are	A reasonable implementation of all member functions (methods) of the class with appropriate handling of the input and return types. Comments to scaffold the code are missing or have ambiguity.	A good implementation of all member functions (methods) of the class with appropriate handling of the input and return types. Comments have been written to scaffold the code	Excellent implementation of all member functions (methods) of the class with appropriate handling of the input and return types. Clear comments have been written to scaffold the code wherever necessary.

		missing or have ambiguity.		wherever necessary but have less clarity.	
1d) Main function's implementation	**2%**	The main function has not been implemented as per the requirement or is not implemented at all. The student has given little to no regard for best practices. Comments are absent or have ambiguity.	The main function has been correctly implemented but in a coarse manner. The student has taken little care to see best practices such as appropriate and consistent variable naming conventions. Comments are absent or have ambiguity.	The main function has been correctly implemented in a concise and professional manner. The student has some care to see best practices such as appropriate and consistent variable naming conventions. Comments are present wherever necessary but have less clarity.	The main function has been correctly implemented in a concise and professional manner. The student has taken great care to see best practices such as appropriate and consistent variable naming conventions. Meaningful comments are present wherever necessary.
2) UML Diagram	**2%**	UML diagram is missing or a partial UML diagram has been drawn with poor use of UML notations and missing many details. A poor understanding of the Location class has been showcased in the diagram.	UML diagram has been drawn using the correct UML notations and appropriately labelled with a few missing details. An adequate understanding of the Location class has been showcased in the diagram.	UML diagram has been drawn using the correct UML notations and appropriately labelled. A good understanding of the Location class has been showcased in the diagram.	UML diagram has been drawn using the correct UML notations and appropriately labelled. An excellent understanding of the Location class has been showcased in the diagram.
3) Process to describe the program including comments	**2%**	Process to describe the program has been documented in an unclear manner. Variable names and naming conventions are neither meaningful nor consistent. Code comments are missing or ambiguous.	Process to describe the program has been documented in a reasonable manner citing the use of meaningful variable and method names, not very consistent naming conventions and inadequate or ambiguous comments.	Process to describe the program has been documented in a good manner citing the use of meaningful variable and method names, consistent naming conventions and comments that have some clarity.	Process to describe the program has been documented in an excellent manner citing the use of meaningful variable and method names, consistent naming conventions and clear comments.
Scenario 2 – RPG Game OOP Design (LO1, LO3). 28%					
Base class creation and implementation	**4%**	Base class has not been implemented or partially implemented with the required constructors, member functions and variables. Student has demonstrated a poor level of understanding of the data types needed for the base class' instance and member (Static) variables.	Base class has been correctly implemented with the required constructors, member functions and variables. Student has demonstrated a reasonable level of understanding of the data types needed for the base class' instance and member (Static) variables.	Base class has been correctly implemented with the required constructors, member functions and variables. Student has demonstrated a good understanding of the data types needed for the base class' instance and member (Static) variables.	Base class has been correctly implemented with the required constructors, member functions and variables. Student has demonstrated an excellent understanding of the data types needed for the base class' instance and member (Static) variables.
Three Derived Classes	**6%**	One or no derived classes have been correctly implemented. Little to no understanding of inheritance and polymorphism concepts has been showcased in the student's code.	At least two derived classes are correctly and completely implemented. A minimal understanding of inheritance and polymorphism concepts has been showcased in the student's code.	All three derived classes are correctly and completely implemented. A good understanding of inheritance and polymorphism concepts has been showcased in the student's code.	All three derived classes are correctly and completely implemented. An excellent understanding of inheritance and polymorphism concepts has been showcased in the student's code.
Main menu and program flow implementation	**16%**	Little to no understanding of	A minimal understanding of program flow and handling of	A good understanding of program flow and handling of	An excellent understanding of program flow and handling of the

		program flow and handling of the main menu is showcased in the student's code. Other functions that facilitate the flow of the main menu are missing or not been appropriately named, inputs and outputs of these functions are poorly handled. Many of the functions required to handle the flow in the main menu are missing or partially implemented.	the main menu is showcased in the student's code. Other functions that facilitate the flow of the main menu have not been appropriately named, implemented with some missing details, input and outputs of these functions are handled with some minor glitches.	the main menu is showcased in the student's code. Other functions that facilitate the flow of the main menu have been appropriately named, implemented with some missing details, input and outputs of these functions are appropriately handled.	main menu is showcased in the student's code. Other functions that facilitate the flow of the main menu have been appropriately named, completely implemented, input and outputs of these functions are appropriately handled.
Use of Best practices	**2%**	Little to negligible understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A minimal understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A good understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	An excellent understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.
Scenario 3 – Shapes calculator (LO1, LO2, LO4) 26%					
UML Diagram	**6%**	UML diagram has been drawn for one or zero classes. UML notations are mostly missing or incorrect. Little to no understanding of access specifiers, inheritance, data members and member functions has been showcased in the diagram.	UML diagram has been drawn using mostly correct UML notations for at least 2 classes. A minimal understanding of access specifiers, inheritance, data members and member functions of at least 2 classes has been showcased in the diagram.	UML diagram has been drawn using the correct UML notations for at least 3 classes. A good understanding of access specifiers, inheritance, data members and member functions of the classes has been showcased in the diagram.	UML diagram has been drawn using the correct UML notations for all 4 classes. An excellent understanding of access specifiers, inheritance, data members and member functions of all classes has been showcased in the diagram.
Implementation of classes	**12%**	A missing or vague class structure with little to no understanding of access specifiers, inheritance hierarchies, program flow, function and loop handling, naming conventions have been showcased. Comments are either missing or ambiguous.	A class structure with less clarity along with a minimal understanding of access specifiers, inheritance hierarchies, program flow, function and loop handling, naming conventions have been showcased. Comments have less clarity	A clear class structure with a good understanding of access specifiers, inheritance hierarchies, program flow, function and loop handling, naming conventions and clear commenting have been showcased.	A clear class structure with an excellent understanding of access specifiers, inheritance hierarchies, program flow, function and loop handling, naming conventions and clear commenting have been showcased.
Appropriate application of function overloading	**4%**	Little to no understanding of function overloading has been demonstrated in the student's code.	A minimal level of understanding of function overloading has been demonstrated via the use of appropriately implemented and invoked functions.	A good understanding of function overloading has been demonstrated via the use of appropriately implemented and invoked functions.	An excellent understanding of function overloading has been demonstrated via the use of appropriately implemented and invoked functions.
Appropriate application of function overriding	**4%**	Little to no understanding of function overriding	A minimal level of understanding of function overriding has been demonstrated via the use of	A good understanding of function overriding has been demonstrated via the use of	An excellent understanding of function overriding has been demonstrated via the use of

		has been demonstrated in the student's code.	appropriately implemented and invoked functions.	appropriately implemented and invoked functions.	appropriately implemented and invoked functions.
Scenario 4 – Alien Game Design (LO1, LO3) 32%					
Implementation of Alien Class	**8%**	Alien class has not been implemented or partially implemented with the required constructors, member functions and variables. Student has demonstrated a poor level of and handling of the data types, instance variables, member functions needed in the Alien class.	Alien class has been correctly implemented with the required constructors, member functions and variables. Student has demonstrated a reasonable level of understanding and handling of the data types, instance variables, member functions needed in the Alien class.	Alien class has been correctly implemented with the required constructors, member functions and variables. Student has demonstrated a good understanding and handling of the data types, instance variables, member functions needed in the Alien class.	Alien class has been correctly implemented with the required constructors, member functions and variables. Student has demonstrated an excellent understanding and handling of the data types, instance variables, member functions needed in the Alien class.
Application of Operator Overloading	**16%**	Little to no understanding of function operator has been demonstrated in the student's code. No or up to two instances of operator overloading specified in the requirement have been implemented.	A minimal level of understanding of operator overloading has been demonstrated via the use of appropriately implemented and invoked functions. At least half the instances of operator overloading specified in the requirement have been correctly implemented.	A good understanding of operator overloading has been demonstrated via the use of appropriately implemented and invoked functions. Most instances of operator overloading specified in the requirement have been correctly implemented.	An excellent understanding of operator overloading has been demonstrated via the use of appropriately implemented and invoked functions. All instances of operator overloading specified in the requirement have been correctly implemented.
Implementation of main function	**6%**	Initialization of the Alien class' objects & comparison with prestige has not been done or done for just a few of cases specified in the requirement The student has demonstrated a poor level of understanding of the requirements via their implementation.	Initialization of the Alien class' objects & comparison with prestige has been correctly done for at least half the cases specified in the requirement The student has demonstrated a reasonable level of understanding of the requirements via their implementation.	Initialization of the Alien class' objects & comparison with prestige has been correctly done for most cases specified in the requirement. The student has demonstrated a good level of understanding of the requirements via their implementation.	Initialization of the Alien class' objects & comparison with prestige has been correctly done for all cases specified in the requirement. The student has demonstrated an excellent level of understanding of the requirements via their implementation.
Use of Best practices	**2%**	A minimal understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A minimal understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A good understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	An excellent understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.
Sum	**100**				