

CS105 – Development Principles-2

Assessment 1

Diploma in Software Development
Yoobee Online

CASE STUDY REPORT

GILBERTO GABON

Author

270204759@yoobeestudent.ac.nz

1. Scenario 1 Solution

a. Task 1: Code for the scenario – contains creation of classes and the output display

```

/*****
* Title : CS-105 Development Principles-2 Assessment 1
* File : yacht.cpp
* Purpose : Scenario1: A program to display the Ocean Race 2021-22 information. This program
showcases Object Oriented Programming (OOP) in C++.
* This program also shows the creation of classes with constructor, encapsulation and inheritance.
* Parameters : N/A
* Returns : N/A
* Author : Gilberto Gabon - Student No.: 270204759
*****/

```

```

#include <iostream>
#include <string>
#include <math.h>

```

```

using namespace std;

```

```

/*****
* Title : CS-105 Development Principles-2 Assessment 1
* Class Name : Location
* Purpose : A class to define a Location object that consists of the following:
* Properties : private int degrees, float minutes, char direction
* : public string latitude = "", longitude = ""
* Methods : public void getpos() --> gets position of the boat and store these into its properties
* Constructor : None
* Returns : N/A
* Author : Gilberto Gabon - Student No.: 270204759
*****/

```

```

class Location
{
    int degrees;
    float minutes;
    char direction;

    public:
        string latitude = "", longitude = "";

        void getpos()
        {

```

```

        char dir;
        cout << "Input the degrees between 0 and 180: ";
        cin >> degrees;
        cout << "Input minutes between 0 and 60: ";
        cin >> minutes;
        cout << "Input direction (E/W/N/S): ";
        cin >> dir;
        direction = toupper(dir); // to convert the direction to uppercase

        if (toupper(direction) == 'N' || toupper(direction) == 'S')
        {
            latitude = to_string(degrees) + '\xF8' + to_string((int)minutes) + "\"" + " " +
            direction + " Latitude";
        }
        else
        {
            longitude = to_string(degrees) + '\xF8' + to_string((int)minutes) + "\"" + " " +
            direction + " Longitude";
        }
    }
};

/***** Title : CS-105
Development Principles-2 Assessment 1
* Class Name : Yacht
* Purpose : A class to define a Yacht object. This class inherits from the 'Location' class. This class of the
following:
* Properties : private int serialNum, objectNum
* Methods : public void get_pos() --> calls the getpos() method from the Location class which this class
inherits
* : public void display() --> displays the yacht's information such as the serial number and its location
* Constructor : public Yacht(int sn, int objnum) --> this receives the serial number of the yacht and the
object count of how many objects (yachts)
* have been created so far
* Returns : N/A
* Author : Gilberto Gabon - Student No.: 270204759
*****/

```

```

class Yacht : public Location
{

    int serialNum, objectNum;

    public:
        Yacht(int sn, int objnum)

```

```

        {
            serialNum = sn;
            objectNum = objnum;
        }
        void get_pos()
        {
            getpos();
        }

        void display()
        {
            cout << "\nThe ship serial number is: " << serialNum << endl;
            cout << "and it's position is: " << latitude << " " << longitude << endl; // the latitude and
            longitude variables here were inherited from Location class
        }
};

int main()
{

    cout << "\n*****Ocean Race 2021-
    22*****\n\n";

    // Define first yacht object
    Yacht yacht1(1, 1);

    // Get information from user for the first yacht object
    cout << "*****" << endl;
    cout << "Enter the Location of the first ship:" << endl;
    yacht1.get_pos();
    yacht1.get_pos();

    // Define second yacht object
    Yacht yacht2(2, 2);

    // Get information from user for the second yacht object
    cout << "*****" << endl;
    cout << "Enter the Location of the second ship:" << endl;
    yacht2.get_pos();
    yacht2.get_pos();

    // Define third yacht object

```

```
Yacht yacht3(3, 3);

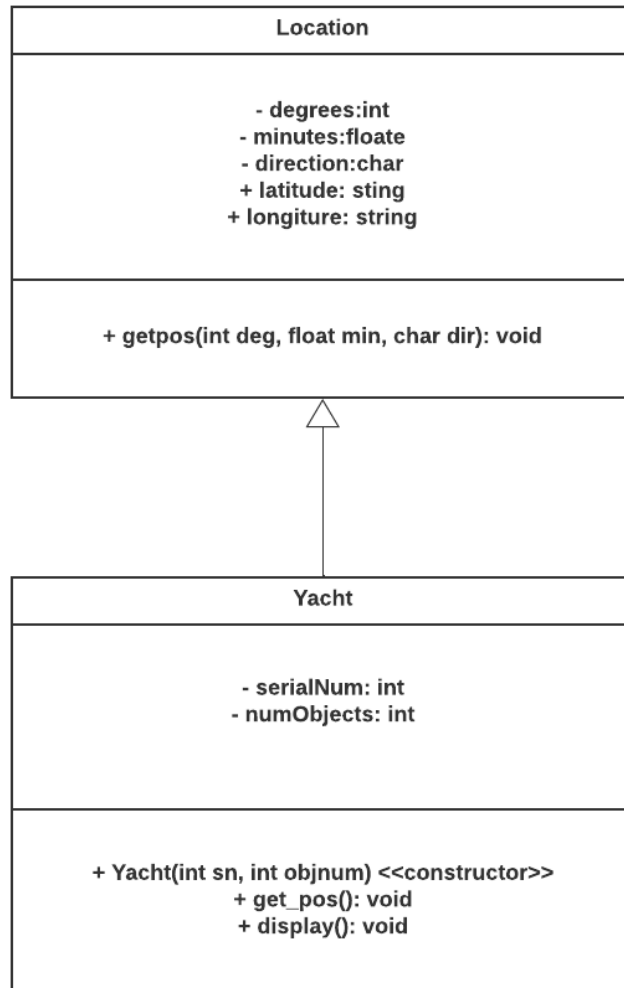
// Get information from user for the third yacht object
cout << "*****" << endl;
cout << "Enter the Location of the third ship:" << endl;
yacht3.get_pos();
yacht3.get_pos();

// Display yacht objects information
cout << "\n\n*****Welcome to Ocean Race 2021-
22*****" << endl;
yacht1.display();
yacht2.display();
yacht3.display();

return 0;
}
```

b. Task 2: UML Diagram**UML Class Diagram - Scenario 1**

[Gilberto Gabon | August 19, 2022]



c. Task 3: Process to solve the problem

i. A **class Location** was created first with the following:

- Properties:
 - Private: These are variables to store location information of the yacht
 - **int degrees;**
 - **float minutes;**
 - **char direction;**
 - Public: These variables are used to store strings for displaying the yacht's location in latitude and longitude format
 - **string latitude = "", longitude = "";**
- Methods:
 - Public:
 - **void getpos()** - this method takes inputs from the user on the yacht's location in degrees, minutes, and direction (E/W/N/S).

Once data are inputted, these are then concatenated into a string and saved in the 'latitude' and 'longitude' string variables.

The 'latitude' string variable stores the location where direction is either North (N) or South (S).

The 'longitude' string variable stores the location where direction is either East (E) or West (W).

ii. Next a **class Yacht** was created. This class inherits the Location class. This class also has a constructor with the following parameters:

1. Yacht(int sn, int objnum): this receives the serial number of the yacht and the object count of how many objects (yachts)

The following are the properties and methods of this class:

- Properties:
 - Private: These are variables to store serial number and object count of the yacht
 - **int serialNum;**

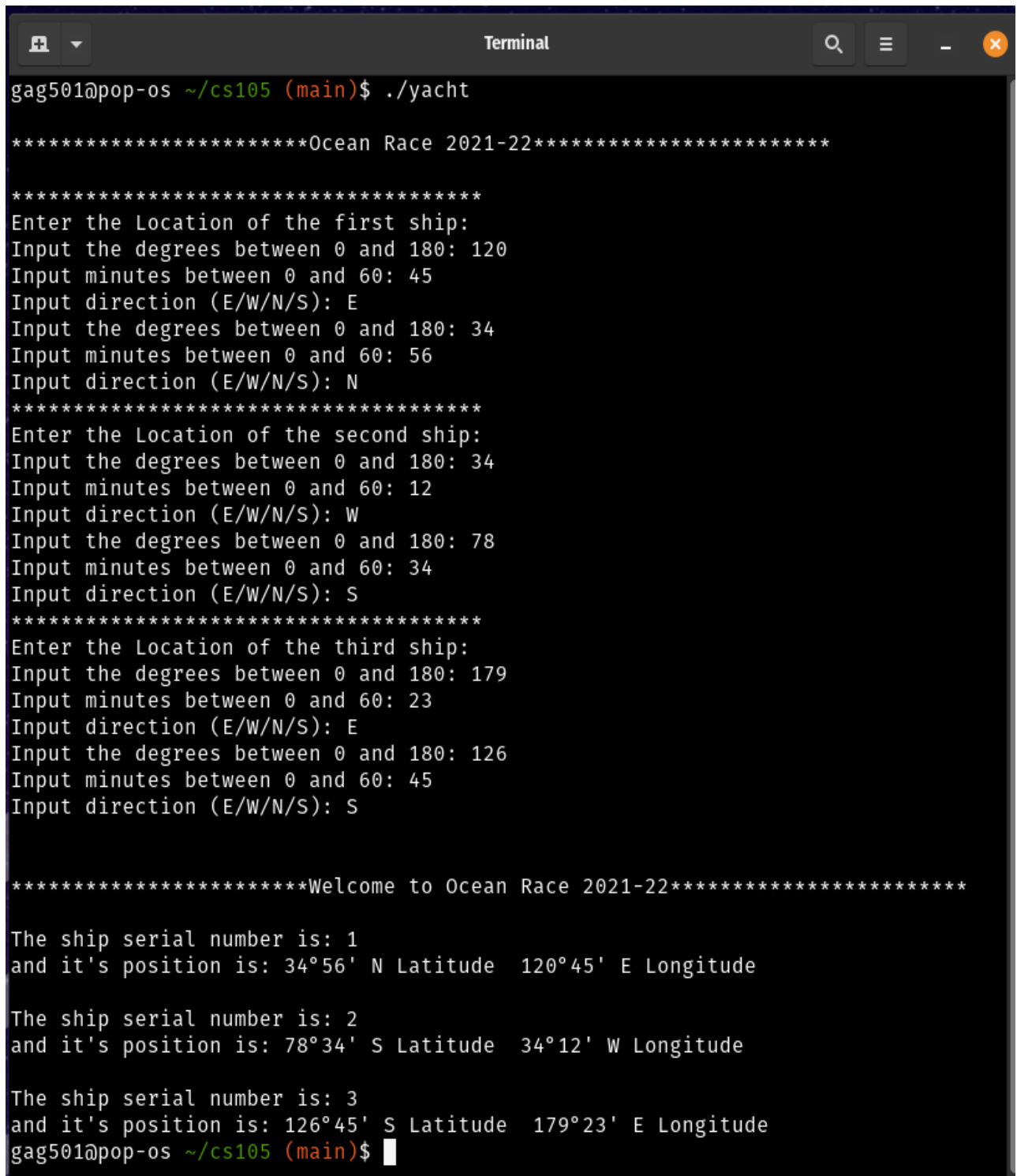
- **int objectNum;**
 - Methods:
 - Public:
 - Constructor **Yacht(int sn, int objnum)**
 - **void get_pos()**: calls the **getpos()** method from the Location class which this class inherits
 - **void display()**: displays the yacht's information such as the serial number and its location.
- iii. Lastly the **main()** function is defined. Inside the main() function, three objects (yachts) are created using the **Yacht class**. Each object of this class contains the properties of both the Location and Yacht classes.

Every creation of the object, the parameters needed by the constructor are passed, such as the “serial number”, and the “object count”.

Once the object is created, the method to get the user input for the boat's location is called: **get_pos()**. This method is called twice, one for the Latitude and another one for the Longitude.

After all user inputs for the three (3) yacht objects, the method to display each object, named **display()**, is called to finally display the yacht's information, such as the serial number and the location.

d. Screenshot



```
gag501@pop-os ~/cs105 (main)$ ./yacht

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

*****
Enter the Location of the first ship:
Input the degrees between 0 and 180: 120
Input minutes between 0 and 60: 45
Input direction (E/W/N/S): E
Input the 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 the degrees between 0 and 180: 34
Input minutes between 0 and 60: 12
Input direction (E/W/N/S): W
Input the 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 the degrees between 0 and 180: 179
Input minutes between 0 and 60: 23
Input direction (E/W/N/S): E
Input the 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 Latitude 120°45' E Longitude

The ship serial number is: 2
and it's position is: 78°34' S Latitude 34°12' W Longitude

The ship serial number is: 3
and it's position is: 126°45' S Latitude 179°23' E Longitude
gag501@pop-os ~/cs105 (main)$
```

2. Scenario 2 Solution

a. Code

```

/*****
* Title : CS-105 Development Principles-2 Assessment 1
* File : rpg.cpp
* Purpose : Scenario 2: A program to create a role-playing game (RPG).This program showcases Object Oriented
Programming (OOP) in C++.
* This program also shows the creation of foundational to represent characters, that is, the entity within the game
that the user of the game can control.
* Parameters : N/A
* Returns : N/A
* Author : Gilberto Gabon - Student No.: 270204759
*****/

#include <string>
#include <cstring>
#include <iostream>
#include <vector>
#include <bits/stdc++.h>
#include "player.h"

using namespace std;

/*****
* Title : CS-105 Development Principles-2 Assessment 1
* Class Name : Player
* Purpose : This is a foundational class for which all characters of the game inherits. This class has the following:
* Properties : private string name, Race race, int hitPoints, int magicPoints;
* Methods : public string getName() --> getter function to get name of the character
* : public Race getRace() --> getter function of type Race (an ENUM defined inside player.h) to get the race of the
character
* : public int getHitPoints() --> getter function to get the hitPoints of the character
* : public int getMagicPoints() --> getter function to get the magicPoints of the character
* : public string whatRace() --> a function to display the string equivalent of the Race enum values
* : public void setName(string n) --> setter function to set the name of the character
* : public void setRace(Race r) --> setter function to set the race of the character
* : public void setHitPoints(int h) --> setter function to set the hitPoints of the character
* : public string attack() --> method to define the attack of the character. This method is overridden in the various
*classes that inherits this base class
*
* Constructor : None
* Returns : N/A
* Author : Gilberto Gabon - Student No.: 270204759
*****/

class Player
{
    string name;
    Race race;
    int hitPoints;
    int magicPoints;

```

```
public:
    string getName()
    {
        transform(name.begin(), name.end(), name.begin(), ::toupper); // return the name in
        uppercase
        return name;
    };

    Race getRace()
    {
        return race;
    };

    int getHitPoints()
    {
        return hitPoints;
    };

    int getMagicPoints()
    {
        return magicPoints;
    };

    string whatRace()
    {
        string raceText = "";
        switch (race)
        {
            case HUMAN:
                raceText = "HUMAN";
                break;
            case ELF:
                raceText = "ELF";
                break;
            case DWARF:
                raceText = "DWARF";
                break;
            case ORC:
                raceText = "ORC";
                break;
            case TROLL:
                raceText = "TROLL";
                break;
        };
        return raceText;
    };

    void setName(string n)
    {
        name = n;
    };
};
```

```

        void setRace(Race r)
        {
            race = r;
        };

        void setHitPoints(int h)
        {
            hitPoints = h;
        };

        void setMagicPoints(int m)
        {
            magicPoints = m;
        };

        string attack()
        {
            return "No attack method defined yet";
        };
};

/*****
* Title : CS-105 Development Principles-2 Assessment 1
* Class Name : Warrior. This class inherits the base class Player
* Purpose : This class defines the Warrior. This inherits the Player class. This class has the following:
* Properties : the ones inherited from Player class
* Methods : the ones inherited from Player class
* : public string attack() --> method to define the attack of the character. This overrides the attack() method of the
Player class
* Constructor : None
* Returns : N/A
* Author : Gilberto Gabon - Student No.: 270204759
*****/
class Warrior : public Player
{
    public:
        string attack()
        {
            return "I will destroy you with my sword, foul demon!";
        };
};

/*****
* Title : CS-105 Development Principles-2 Assessment 1
* Class Name : Priest. This class inherits the base class Player
* Purpose : This class defines the Warrior. This inherits the Player class. This class has the following:
* Properties : the ones inherited from Player class
* Methods : the ones inherited from Player class
* : public string attack() --> method to define the attack of the character. This overrides the attack() method of the
Player class
* Constructor : None
* Returns : N/A

```

* Author : Gilberto Gabon - Student No.: 270204759

*****/

```
class Priest : public Player
{
```

```
    public:
        string attack()
        {
            return "I will assault you with my holy wrath!";
        };
};
```

*****/

* Title : CS-105 Development Principles-2 Assessment 1

* Class Name : Mage. This class inherits the base class Player

* Purpose : This class defines the Warrior. This inherits the Player class. This class has the following:

* Properties : the ones inherited from Player class

* Methods : the ones inherited from Player class

* : public string attack() --> method to define the attack of the character. This overrides the attack() method of the Player class

* Constructor : None

* Returns : N/A

* Author : Gilberto Gabon - Student No.: 270204759

*****/

```
class Mage : public Player
{
```

```
    public:
        string attack()
        {
            return "I will crush you with my arcane missiles!";
        };
};
```

```
int main()
```

```
{
```

```
    // Define the vectors for the different classes (warrior, priest, mage)
```

```
    vector<Warrior> warrior;
```

```
    vector<Priest> priest;
```

```
    vector<Mage> mage;
```

```
    // Ask the user to create the character. The program loops until the user decides to finish the character
    creation by selecting '4' in the menu
```

```
    int ch = 0;
```

```
    int raceType;
```

```
    string charName;
```

```
    while (ch != 4)
```

```
    {
```

```
        cout << "\nCHARACTER CREATION" << endl;
```

```
        cout << "1. Create a Warrior" << endl;
```

```
        cout << "2. Create a Priest" << endl;
```

```
        cout << "3. Create a Mage" << endl;
```

```
cout << "4. Finish creating player characters" << endl;
cout << "Choice: ";
cin >> ch;
if (ch != 4)
{
    cout << "\nWhat race do you want?\n";
    cout << "1. Human\n";
    cout << "2. Elf\n";
    cout << "3. Dwarf\n";
    cout << "4. Orc\n";
    cout << "5. Troll\n";
    cout << "Choice: ";
    cin >> raceType;

    cout << "\nWhat would you like to name your character? ";
    cin >> charName;

    // Depending on the choice, each character is added to its own vector of characters (e.g. warrior, priest,
    mage)
    switch (ch)
    {
    case 1:
    {
        Warrior w;
        w.setName(charName);
        w.setRace((Race)raceType);
        w.setHitPoints(200);
        w.setMagicPoints(0);

        warrior.push_back(w); // add new warrior to the list of warrior characters

        break;
    }

    case 2:
    {
        Priest p;
        p.setName(charName);
        p.setRace((Race)raceType);
        p.setHitPoints(100);
        p.setMagicPoints(200);

        priest.push_back(p); // add new priest to the list of priest characters

        break;
    }

    case 3:
    {
        Mage m;
        m.setName(charName);
        m.setRace((Race)raceType);
```

```

        m.setHitPoints(200);
        m.setMagicPoints(0);

        mage.push_back(m); // add new mage to the list of mage characters

        break;
    }

    default:
        break;
    }
}

// Display the characters according to their classes
cout << "\n\n-----";
cout << "\nWARRIORS LIST: ";
cout << "\n-----";
for (int i = 0; i < (int)warrior.size(); i++)
{
    cout << "\nI am a WARRIOR with name " << warrior[i].getName() << " and with race " <<
    warrior[i].whatRace() << " and my attack is: " << warrior[i].attack();
}
cout << "\n\n-----";
cout << "\nPRIESTS LIST: ";
cout << "\n-----";
for (int i = 0; i < (int)priest.size(); i++)
{
    cout << "\nI am a PRIEST with name " << priest[i].getName() << " and with race " << priest[i].whatRace()
    << " and my attack is: " << priest[i].attack();
}
cout << "\n\n-----";
cout << "\nMAGE LIST: ";
cout << "\n-----";
for (int i = 0; i < (int)mage.size(); i++)
{
    cout << "\nI am a MAGE with name " << mage[i].getName() << " and with race " << mage[i].whatRace() <<
    " and my attack is: " << mage[i].attack();
}
cout << endl;

return 0;
}

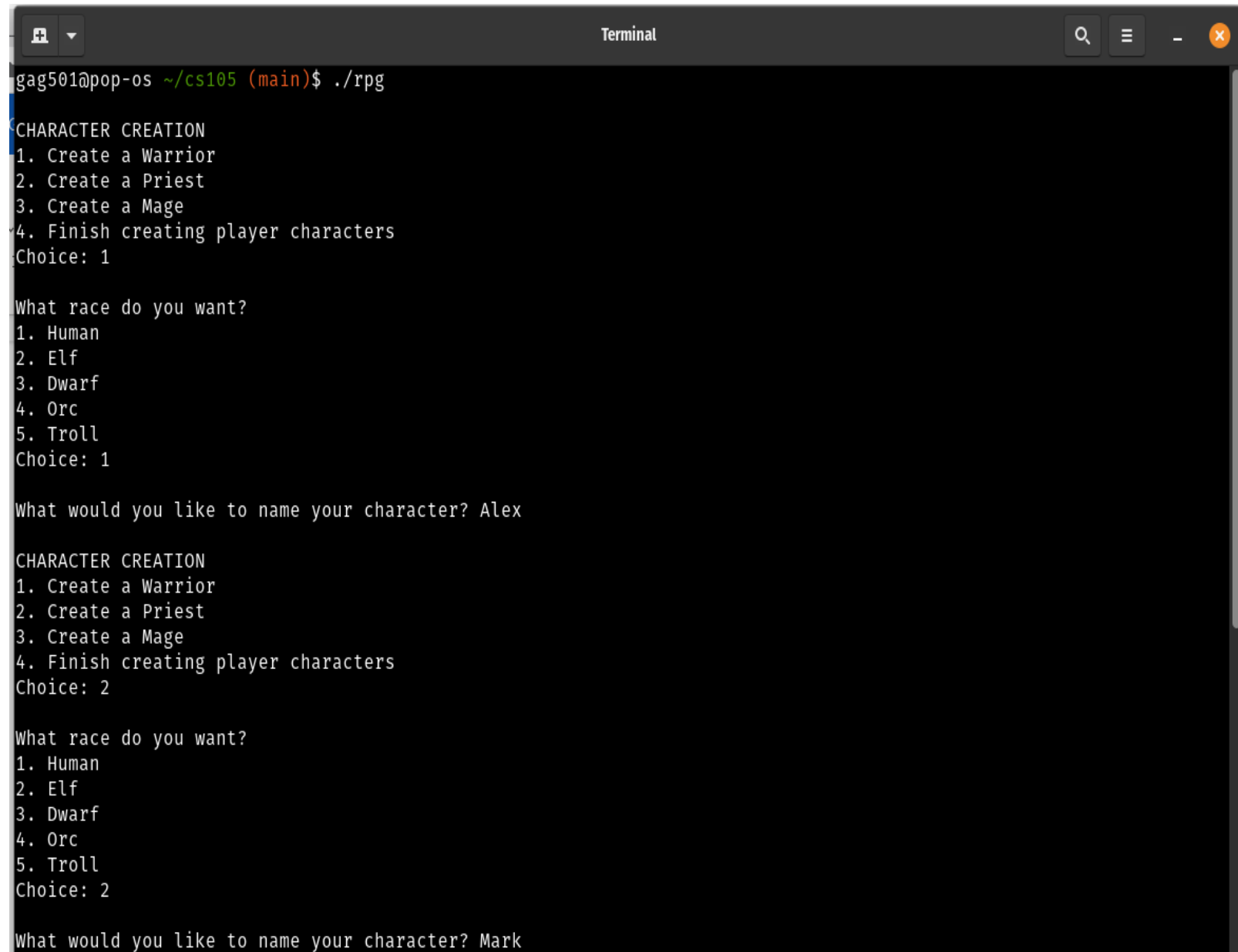
```

b. Process to solve the problem

- i. A foundation (base) **class Player** was created with the following properties and methods:
 - Properties:
 - a. **private string name, Race race, int hitPoints, int magicPoints;**
 - Methods:
 - a. **public string getName():** getter function to get name of the character
 - b. **public Race getRace():** getter function of type Race (*an ENUM defined inside player.h*) to get the race of the character
 - c. **public int getHitPoints():** getter function to get the hitPoints of the character
 - d. **public int getMagicPoints():** getter function to get the magicPoints of the character
 - e. **public string whatRace():** a function to display the string equivalent of the Race enum values
 - f. **public void setName(string n):** setter function to set the name of the character
 - g. **public void setRace(Race r):** setter function to set the race of the character
 - h. **public void setHitPoints(int h):** setter function to set the hitPoints of the character
 - i. **public string attack():** method to define the attack of the character. This method is overridden in the various classes that inherits this base class
- ii. Next, derived classes for the various characters were created. These classes are inherited from the Player class. The only function that is overridden for each derived class is the attack() function. The derived classes are named Warrior, Priest, and Mage.

- iii. Finally, the main() function is created. Inside the main() function the following are done:
- Creation of character vectors for each character type (e.g. Warrior, Priest, and Mage)
 - User is then prompted with a menu to create the different characters (i.e., Warrior, Priest, and Mage). The program makes a loop to ask for menu choices until the user selects the menu **'4. Finish creating player characters'**. Once this is selected, the program exits from the loop.
 - Every time a user creates a character, he/she is asked for the race of the character as well as the name of the character.
 - Once all the needed information for the character is entered, the character is then added to the vector variable of that character using the **push_back()** function.
 - After all the creation of the characters, the program then displays the list of the characters created grouped according to the character type, i.e., Warrior, Priest, and Mage
 - The program makes use of looping (for loop) in displaying the contents of each vector variable.

c. Screenshots

A terminal window titled "Terminal" with a dark background and light text. The prompt is "gag501@pop-os ~/cs105 (main)\$". The user has run the command "./rpg". The program displays a menu for "CHARACTER CREATION" with four options: "1. Create a Warrior", "2. Create a Priest", "3. Create a Mage", and "4. Finish creating player characters". The user has entered "Choice: 1". The program then asks "What race do you want?" and lists five options: "1. Human", "2. Elf", "3. Dwarf", "4. Orc", and "5. Troll". The user has entered "Choice: 1". The program then asks "What would you like to name your character?" and the user has entered "Alex". The program then displays the same "CHARACTER CREATION" menu again. The user has entered "Choice: 2". The program then asks "What race do you want?" and lists the same five options. The user has entered "Choice: 2". The program then asks "What would you like to name your character?" and the user has entered "Mark".

```
gag501@pop-os ~/cs105 (main)$ ./rpg
CHARACTER CREATION
1. Create a Warrior
2. Create a Priest
3. Create a Mage
4. Finish creating player characters
Choice: 1
What race do you want?
1. Human
2. Elf
3. Dwarf
4. Orc
5. Troll
Choice: 1
What would you like to name your character? Alex
CHARACTER CREATION
1. Create a Warrior
2. Create a Priest
3. Create a Mage
4. Finish creating player characters
Choice: 2
What race do you want?
1. Human
2. Elf
3. Dwarf
4. Orc
5. Troll
Choice: 2
What would you like to name your character? Mark
```

```
Terminal
CHARACTER CREATION
1. Create a Warrior
2. Create a Priest
3. Create a Mage
4. Finish creating player characters
Choice: 3

What race do you want?
1. Human
2. Elf
3. Dwarf
4. Orc
5. Troll
Choice: 3

What would you like to name your character? Raghii

CHARACTER CREATION
1. Create a Warrior
2. Create a Priest
3. Create a Mage
4. Finish creating player characters
Choice: 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 my Holy Wrath!

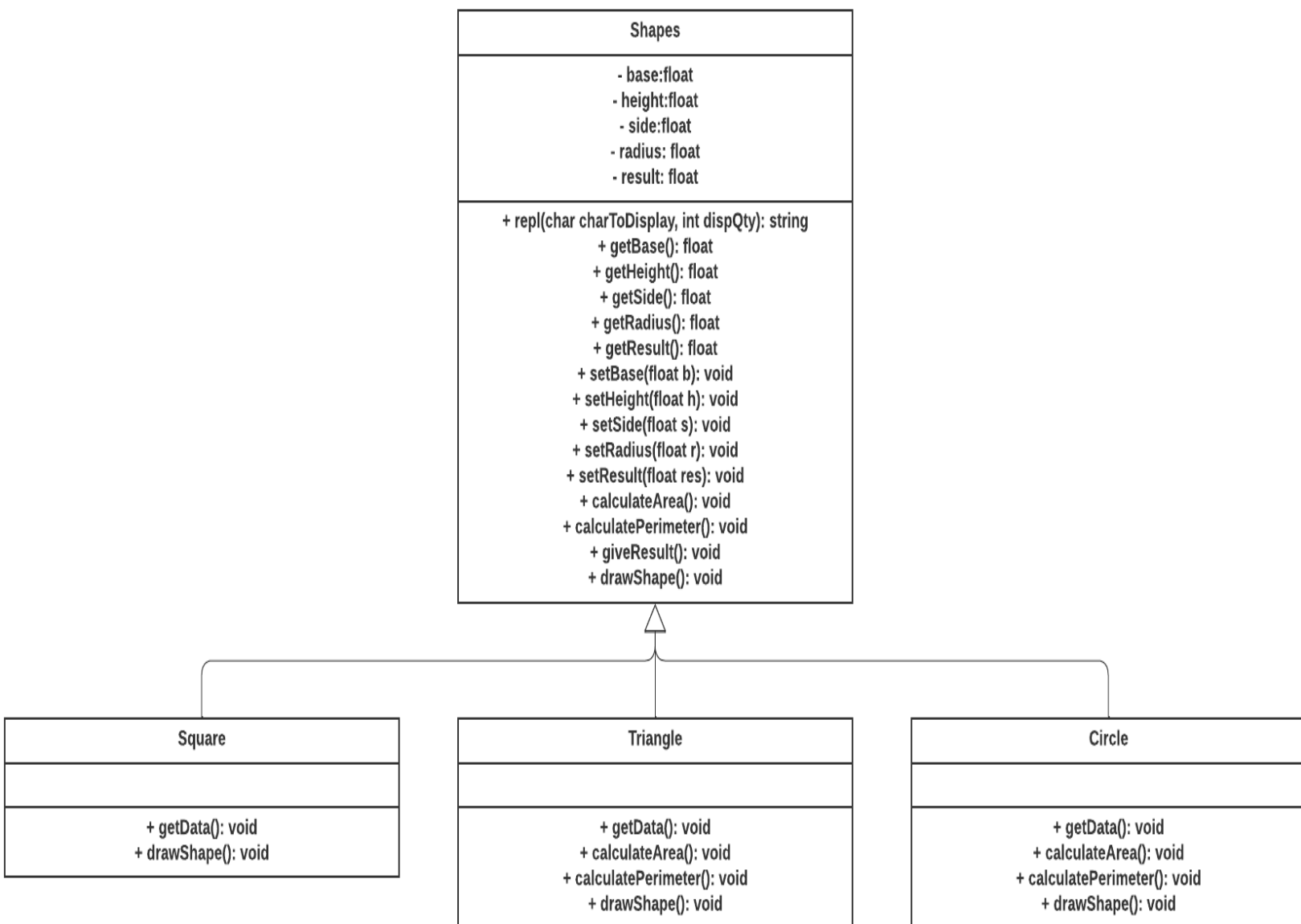
-----
MAGE LIST:
-----
I am a MAGE with name RAGHII and with race DWARF and my attack is: I will crush you with my arcane missiles!
gag501@pop-os ~/cs105 (main)$
```

3. Scenario 3 Solution

a. Task1: UML Diagram

UML Class Diagram - Scenario 3

[Gilberto Gabon | August 20, 2022]



b. Task2: Code

```

/*****
* Title      : CS-105 Development Principles-2 Assessment 1
* File       : shapes.cpp
* Purpose    : Scenario 3: A program to create shapes and display their areas and perimeter values. This
               program showcases Object Oriented Programming (OOP) in C++. This program showcases
               classes creation, inheritance, encapsulation, and polymorphism through function overriding
* Parameters : N/A
* Returns    : N/A
* Author     : Gilberto Gabon - Student No.: 270204759
*****/

#include <iostream>
#include <string>
#include <math.h>
#include <vector>
#include "gglib.cpp" //own library of functions

using namespace std;

/*****
* Title      : CS-105 Development Principles-2 Assessment 1
* Class Name : Shapes
* Purpose    : This is a foundational (base) class for which all types of shapes inherits. This class has the
               following:
*
               Properties : private float base = 0.0, height = 0.0, side = 0.0, radius = 0.0, result = 0.0;
*
               Methods   : public string repl(char charToDisplay, int dispQty) --> general purpose function
               to display a series of characters - sort of replicating the display. This is useful for displaying
               lines.
*
               : public float getBase() --> getter function to get the base of the shape
*
               : public float getHeight() --> getter function to the height of the shape
*
               : public float getSide() --> getter function to get side of the shape
*
               : public float getRadius() --> getter function to get the radius of the circle
*
               : public float getResult() --> getter function to return the value of the 'result' property
*
               : public void setBase(float b) --> setter function to set the value of the base of the shape
*
               : public void setHeight(float h) --> setter function to set the value of the height of the shape
*
               : public void setSide(float s) --> setter function to set the value of the height of the shape
*
               : public void setRadius(float r) --> setter function to set the value of the radius of the circle
*
               : public void calculateArea() --> method to calculate the area of the shape. Initial computation is
               based on the shape of a rectangle
*
               : public void calculatePerimeter() --> method to calculate the perimeter of the shape. Initial
               computation is based on the shape of a rectangle
*
               : public void getData() --> method to get the dimension of the shape. Initial computation is based
               on the shape of a rectangle
*
               : public void giveResult() --> method to display the value of the 'result' property. This property is
               updated during the computation of either area or perimeter
*****/

```

* : public void drawShape() --> method to draw the shape. Initial drawing is based on the shape of a rectangle

* Constructor : None

* Returns : N/A

* Author : Gilberto Gabon - Student No.: 270204759

*****/

class Shapes

```
{
private:
    float base = 0.0, height = 0.0, side = 0.0, radius = 0.0, result = 0.0;
```

```
public:
    string repl(char charToDisplay, int dispQty)
    {
        string returnedString = "";
        for (int i = 0; i < dispQty; i++)
        {
            returnedString.push_back(charToDisplay);
        }
        return returnedString;
    }
```

// Define getters

```
float getBase()
```

```
{
    return base;
}
```

```
float getHeight()
```

```
{
    return height;
}
```

```
float getSide()
```

```
{
    return side;
}
```

```
float getRadius()
```

```
{
    return radius;
}
```

```
float getResult()
```

```
{
    return result;
}
```

// Define setters

```
void setBase(float b)
```

```
{
    base = b;
}
void setHeight(float h)
{
    height = h;
}
void setSide(float s)
{
    side = s;
}
void setRadius(float r)
{
    radius = r;
}

void setResult(float res)
{
    result = res;
}

// Default function - based on a rectangle

void calculateArea()
{
    result = base * height;
}

void calculatePerimeter()
{
    result = 2 * (base + height);
}

void getData()
{
    cout << " \nEnter base (cm): ";
    cin >> base;
    cout << "Enter height (cm): ";
    cin >> height;
}

void giveResult()
{
    cout << "The result is: " << result;
}

void drawShape()
```

```

{
    // base shape is a rectangle
    cout << endl;
    for (int y = 1; y <= height / 2; y++)
    {
        cout << repl('.', base) << endl;
    }
    cout << endl;
}
};

/*****
* Title      : CS-105 Development Principles-2 Assessment 1
* Class Name : Square. This inherits the Shapes class
* Purpose    : This is a derived class that inherits the Shapes class
* Properties : as inherited from Shapes class
* Methods    : as inherited from Shapes class plus
*            : public void getData() --> overrides the getData() method from the Shapes class. This metho
                now only asks for the side of the shape.
*            : public void drawShape() --> overrides the drawShape() method from the Shapes class. This
                draws the square shape.
* Constructor : None
* Returns     : N/A
* Author      : Gilberto Gabon - Student No.: 270204759
*****/

class Square : public Shapes
{
public:
    void getData()
    {
        float side;
        cout << " \nEnter side (cm): ";
        cin >> side;
        setBase(side); // update the base
        setHeight(side); // update the height
    }

    void drawShape()
    {
        cout << endl;
        for (int y = 1; y <= getHeight(); y++)
        {
            cout << repl('.', getBase() * 2) << endl;
        }
        cout << endl;
    }
}

```



```

}
};

```

```

/*****
* Title      : CS-105 Development Principles-2 Assessment 1
* Class Name : Triangle. This inherits the Shapes class
* Purpose    : This is a derived class that inherits the Shapes class
* Properties : as inherited from Shapes class
* Methods    : as inherited from Shapes class plus
*            : public void getData() --> overrides the getData() method from the Shapes class. This method
                asks for an additional input of the side of the triangle.
*            : public void calculateArea() --> overrides the calculateArea() method from the Shapes class. This
                calculates the area of a triangle
*            : public void calculatePerimeter() --> overrides the calculatePerimeter() method from the
                Shapes class. This calculates the perimeter of a triangle
*            : public void drawShape() --> overrides the drawShape() method from the Shapes class. This
                draws the triangle shape.
* Constructor : None
* Returns     : N/A
* Author      : Gilberto Gabon - Student No.: 270204759
*****/

```

```

class Triangle : public Shapes

```

```

{

```

```

public:

```

```

    void getData()

```

```

    {

```

```

        float base, height, side;

```

```

        cout << " \nEnter base (cm): ";

```

```

        cin >> base;

```

```

        cout << "Enter height (cm): ";

```

```

        cin >> height;

```

```

        cout << "Enter side (cm): ";

```

```

        cin >> side;

```

```

        setBase(base); // update base

```

```

        setHeight(height); // update height

```

```

        setSide(side); // update side

```

```

    }

```

```

    void calculateArea() // overloading this function - area of the triangle

```

```

    {

```

```

        float result;

```

```

        float b = getBase();

```

```

        float h = getHeight();

```

```

        result = 0.5 * b * h; // A = 1/2 * base * height

```

```

        setResult(result); // update result based on this formula for the triangle

```

```

    }

```

```

void calculatePerimeter() // overloading this function - area of the triangle
{
    float result;
    float b = getBase();
    float h = getHeight();
    float s = getSide();
    result = b + h + s; // P = base + height + side
    setResult(result); // update result based on this formula for the triangle
}

void drawShape()
{

    int multiplier = 0;
    multiplier = getBase() / getHeight();
    cout << "\n." << endl;
    for (int y = 1; y <= getHeight(); y++)
    {

        cout << repl('.', multiplier * y) << endl;
    }
    cout << endl;
}
};

/*****
* Title      : CS-105 Development Principles-2 Assessment 1
* Class Name : Circle. This inherits the Shapes class
* Purpose    : This is a derived class that inherits the Shapes class
* Properties : as inherited from Shapes class
* Methods    : as inherited from Shapes class plus
*            : public void getData() --> overrides the getData() method from the Shapes class. This method
                asks the radius of the circle.
*            : public void calculateArea() --> overrides the calculateArea() method from the Shapes class. This
                calculates the area of the circle
*            : public void calculatePerimeter() --> overrides the calculatePerimeter() method from the
                Shapes class. This calculates the perimeter of the circle
*            : public void drawShape() --> overrides the drawShape() method from the Shapes class. This
                draws the circle shape.
* Constructor : None
* Returns     : N/A
* Author      : Gilberto Gabon - Student No.: 270204759
*****/

class Circle : public Shapes
{

public:
    void getData()

```

```
{
    float r;
    cout << "\nEnter radius (cm): ";
    cin >> r;
    setRadius(r);
}

void calculateArea() // overloading this function - area of a circle
{
    float result;
    float r = getRadius();
    result = 3.14159 * r * r; // A = pi * radius * radius
    setResult(result); // update result based on this formula for the circle
}

void calculatePerimeter() // overloading this function - area of a circle
{
    float result;
    float r = getRadius();
    result = 2 * 3.14159 * r; // C = 2 * pi * radius
    setResult(result); // update result based on this formula for the circle
}

void drawShape()
{
    float r = getRadius();
    const int SIZE = 2 * r; // diameter of the circle

    char canvas[SIZE][SIZE]; // size of the drawing canvass = diameter of the circle

    // Initialize contents of the array with blanks
    for (int i = 0; i < SIZE; i++)
    {
        for (int j = 0; j < SIZE; j++)
        {
            canvas[i][j] = ' '; // initialize with SPACE character (blank)
        }
    }

    for (int row = 0; row < SIZE; row++)
    {
        for (int col = 0; col < SIZE; col++)
        {
            int x = col - SIZE / 2;
            int y = SIZE / 2 - row;

            int r2 = x * x + y * y; // plotting equation of circle  $r^2 = x^2 + y^2$ 
```

```

        if (r2 < (r * r))
        {
            canvas[row][col] = '.'; // any value that is within the square of the radius, save a dot (.)
        }
    }
}

// print contents of the array to draw the circle
for (int row = 0; row < SIZE; row++)
{
    for (int col = 0; col < SIZE; col++)
    {
        cout << canvas[row][col] << " ";
    }

    cout << endl;
}
cout << endl;
}
};

// Functions declarations
void showSquare();
void showRectangle();
void showTriangle();
void showCircle();
void showMainMenu();

// main function
int main()
{
    showMainMenu();
    return 0;
}

/*****
* Title      : CS-105 Development Principles-2 Assessment 1
* Function Name: showSquare()
* Purpose    : Function to compute the area and perimeter of a square. This also draws the square shape
               based on the inputted dimensions.
* Parameters : None
* Returns    : None
* Author     : Gilberto Gabon - Student No.: 270204759
*****/

void showSquare()
{

```

```

Square s;
int choice = 0;
vector<string> menu = {
    "\n=====",
    "    Square Calculator",
    "=====",
    "1. Area (Area = base * base sq. units)",
    "2. Perimeter (Perimeter = 4 * base units)",
    "3. Go back to main menu (Shapes Calculator)",
    "=====",
    ""};

while (choice != 3)
{
    choice = showMenu(menu);
    if (choice != 3)
    {
        s.getData();
        s.drawShape();
    }

    switch (choice)
    {
    case 1:
    {
        s.calculateArea();
        cout << "Area of square. ";
        s.giveResult();
        cout << " sq.cm.\n";
    }
    break;
    case 2:
    {
        s.calculatePerimeter();
        cout << "Perimeter of square. ";
        s.giveResult();
        cout << " cm.\n";
    }
    break;

    default:
        break;
    }
}
}

```

```

/*****
* Title      : CS-105 Development Principles-2 Assessment 1
* Function Name: showRectangle()
* Purpose    : Function to compute the area and perimeter of a rectangle. This also draws the rectangle
               shape based on the inputted dimensions.
* Parameters : None
* Returns    : None
* Author     : Gilberto Gabon - Student No.: 270204759
*****/

```

```

void showRectangle()
{
    Shapes r;

    int choice = 0;
    vector<string> menu = {
        "\n===== ",
        "    Rectangle Calculator",
        "===== ",
        "1. Area (Area = base * height sq. units)",
        "2. Perimeter (Perimeter = 2 * (base + height) units)",
        "3. Go back to main menu (Shapes Calculator)",
        "===== ",
        ""};

    while (choice != 3)
    {
        choice = showMenu(menu);
        if (choice != 3)
        {
            r.getData();
            r.drawShape();
        }

        switch (choice)
        {
            case 1:
            {
                r.calculateArea();
                cout << "Area of rectangle. ";
                r.giveResult();
                cout << " sq.cm.\n";
            }
            break;
            case 2:
            {
                r.calculatePerimeter();
                cout << "Perimeter of rectangle. ";
            }
        }
    }
}

```

```

        r.giveResult();
        cout << " cm.\n";
    }
    break;

    default:
        break;
    }
}
}

/*****
* Title      : CS-105 Development Principles-2 Assessment 1
* Function Name: showTriangle()
* Purpose    : Function to compute the area and perimeter of a triangle. This also draws the triangle shape
               based on the inputted dimensions.
* Parameters : None
* Returns    : None
* Author     : Gilberto Gabon - Student No.: 270204759
*****/

void showTriangle()
{
    Triangle t;

    int choice = 0;
    vector<string> menu = {
        "\n===== ",
        "    Triangle Calculator",
        "===== ",
        "1. Area (Area = 1/2 * base * height sq. units)",
        "2. Perimeter (Perimeter = 2 * (base + height) units)",
        "3. Go back to main menu (Shapes Calculator)",
        "===== ",
        ""};

    while (choice != 3)
    {
        choice = showMenu(menu);
        if (choice != 3)
        {
            t.getData();
            t.drawShape();
        }

        switch (choice)
        {
            case 1:
            {

```

```

        t.calculateArea();
        cout << "Area of triangle. ";
        t.giveResult();
        cout << " sq.cm.\n";
    }
    break;
case 2:
{
    t.calculatePerimeter();
    cout << "Perimeter of triangle. ";
    t.giveResult();
    cout << " cm.\n";
}
break;

default:
    break;
}
}
}

/*****
* Title      : CS-105 Development Principles-2 Assessment 1
* Function Name: showCircle()
* Purpose    : Function to compute the area and perimeter of a circle. This also draws the circle shape based
               on the inputted dimensions.
* Parameters : None
* Returns   : None
* Author    : Gilberto Gabon - Student No.: 270204759
*****/
void showCircle()
{
    Circle c;

    int choice = 0;
    vector<string> menu = {
        "\n===== ",
        "    Circle Calculator",
        "===== ",
        "1. Area (A = pi * radius * radius sq. units)",
        "2. Circumference (C = 2 * pi * radius units)",
        "3. Go back to main menu (Shapes Calculator)",
        "===== ",
        ""};

    while (choice != 3)
    {

```



```

choice = showMenu(menu);
if (choice != 3)
{
    c.getData();
    c.drawShape();
}

switch (choice)
{
case 1:
{
    c.calculateArea();
    cout << "Area of circle. ";
    c.giveResult();
    cout << " sq.cm.\n";
}
break;
case 2:
{
    c.calculatePerimeter();
    cout << "Circumference of circle. ";
    c.giveResult();
    cout << " cm.\n";
}
break;

default:
    break;
}
}
}

/*****
* Title      : CS-105 Development Principles-2 Assessment 1
* Function Name: showMainMenu()
* Purpose    : Function to show the Main Menu of the program
* Parameters : None
* Returns    : None
* Author     : Gilberto Gabon
*****/

void showMainMenu()
{

    int choice = 0;
    vector<string> menu = {
        "=====",
        " Shapes Calculator",
        "====="
    }

```

```
"1. Square",  
"2. Rectangle",  
"3. Triangle",  
"4. Circle",  
"5. Exit",  
"=====",  
""};
```

```
while (choice != 5)  
{  
    choice = showMenu(menu);  
  
    switch (choice)  
    {  
        case 1:  
            showSquare();  
            break;  
        case 2:  
            showRectangle();  
            break;  
        case 3:  
            showTriangle();  
            break;  
        case 4:  
            showCircle();  
            break;  
  
        default:  
            break;  
    }  
}  
}
```

c. **Process to solve the problem**

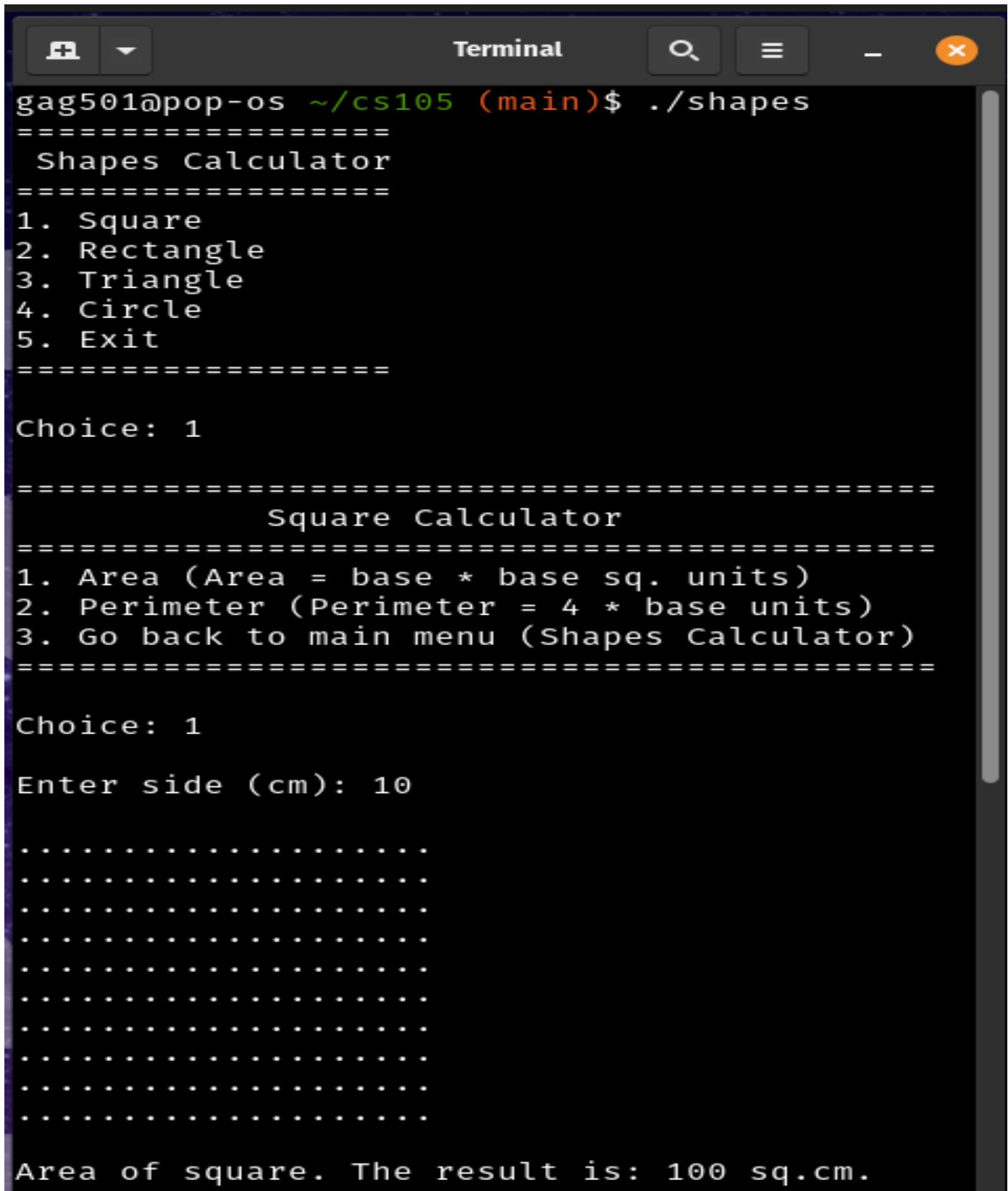
i. Base **class Shapes** was created. This class contains the following properties and methods:

- Properties:
 - a. private: float base = 0.0, height = 0.0, side = 0.0, radius = 0.0, result = 0.0;
- Methods:
 - a. public string repl(char charToDisplay, int dispQty) --> general purpose function to display a series of characters - sort of replicating the display. This is useful for displaying lines.
 - b. public float getBase() --> getter function to get the base of the shape
 - c. public float getHeight() --> getter function to the height of the shape
 - d. public float getSide() --> getter function to get side of the shape
 - e. public float getRadius() --> getter function to get the radius of the circle
 - f. public float getResult() --> getter function to return the value of the 'result' property
 - g. public void setBase(float b) --> setter function to set the value of the base of the shape
 - h. public void setHeight(float h) --> setter function to set the value of the height of the shape
 - i. public void setSide(float s) --> setter function to set the value of the height of the shape

- j. `public void setRadius(float r)` --> setter function to set the value of the radius of the circle
 - k. `public void calculateArea()` --> method to calculate the area of the shape. Initial computation is based on the shape of a rectangle
 - l. `public void calculatePerimeter()` --> method to calculate the perimeter of the shape. Initial computation is based on the shape of a rectangle
 - m. `public void getData()` --> method to get the dimension of the shape. Initial computation is based on the shape of a rectangle
 - n. `public void giveResult()` --> method to display the value of the 'result' property. This property is updated during the computation of either area or perimeter
 - o. `public void drawShape()` --> method to draw the shape. Initial drawing is based on the shape of a rectangle
- ii. Then derived classes were created, such as Square, Triangle, and Circle. All these classes inherit the base class Shapes. In each of this class, there are methods that are overridden according to the specific requirement of the class. Example is the `getData()` method in the Triangle class. This method added the input from the user for the side of the triangle in addition to the default width, and height properties that were asked from the user.
- iii. Finally, the `main()` function is created. Inside the `main()` function, the `showMainMenu()` is called. The function `showMainMenu()` runs a loop showing the list of menus for the user to choose. The following are the choices:
- "1. Square",

- "2. Rectangle",
 - "3. Triangle",
 - "4. Circle",
 - "5. Exit",
- iv. For each choice, a subsequent sub-menu is shown, asking the user to choose to either compute the area, compute the perimeter, or go back to the previous menu.
- v. Once the user choose the Exit menu, the program will end.

d. Screenshots



```
gag501@pop-os ~/cs105 (main)$ ./shapes
=====
Shapes Calculator
=====
1. Square
2. Rectangle
3. Triangle
4. Circle
5. Exit
=====

Choice: 1

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

Choice: 1

Enter side (cm): 10

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Area of square. The result is: 100 sq.cm.
```

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

Choice: 2

Enter side (cm): 10

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Perimeter of square. The result is: 40 cm.
```

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

Choice: 3
=====
Shapes Calculator
=====
1. Square
2. Rectangle
3. Triangle
4. Circle
5. Exit
=====

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

Choice: 1

Enter base (cm): 30
Enter height (cm): 10

.....
.....
.....
.....
.....

Area of rectangle. The result is: 300 sq.cm.
```



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

Choice: 2

Enter base (cm): 30
Enter height (cm): 10

.....
.....
.....
.....
.....

Perimeter of rectangle. The result is: 80 cm.
```

```
Terminal

Choice: 3
=====
  Shapes Calculator
=====
1. Square
2. Rectangle
3. Triangle
4. Circle
5. Exit
=====

Choice: 3

=====
      Triangle Calculator
=====
1. Area (Area =  $1/2 * \text{base} * \text{height sq. units}$ )
2. Perimeter (Perimeter =  $2 * (\text{base} + \text{height}) \text{ units}$ )
3. Go back to main menu (Shapes Calculator)
=====

Choice: 1

Enter base (cm): 30
Enter height (cm): 15
Enter side (cm): 40
```

```
Terminal
```

```
Triangle Calculator  
=====
```

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

```
Choice: 1
```

```
Enter base (cm): 30  
Enter height (cm): 15  
Enter side (cm): 40
```

```
.  
. .  
. . . .  
. . . . .  
. . . . . .  
. . . . . . .  
. . . . . . . .  
. . . . . . . . .  
. . . . . . . . . .  
. . . . . . . . . . .  
. . . . . . . . . . . .  
. . . . . . . . . . . . .  
. . . . . . . . . . . . . .  
. . . . . . . . . . . . . . .  
. . . . . . . . . . . . . . . .  
. . . . . . . . . . . . . . . . .  
. . . . . . . . . . . . . . . . . .  
. . . . . . . . . . . . . . . . . . .  
. . . . . . . . . . . . . . . . . . . .  
. . . . . . . . . . . . . . . . . . . . .
```

```
Area of triangle. The result is: 225 sq.cm.
```

```
=====
```

```
Triangle Calculator  
=====
```

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

```
Choice:
```

[illegible]

[illegible]

4. Scenario 4 Solution

a. Code

```

/*****

* Title      : CS-105 Development Principles-2 Assessment 1
* File       : aliens.cpp
* Purpose    : Scenario 4: A program to create a game universe for an Alien based game. This program
               showcases Object Oriented Programming (OOP) in C++. This program showcases class
               creation with constructor, encapsulation, and polymorphism through operator
               overloading
* Parameters: N/A
* Returns    : N/A
* Author     : Gilberto Gabon - Student No.: 270204759
*****/

#include <iostream>
#include <string>
#include <stdlib.h>
#include <time.h>
#include "gglib.cpp" //own library of functions

using namespace std;

/*****

* Title      : CS-105 Development Principles-2 Assessment 1
* Class Name  : Alien
* Purpose    : This is a class for creating Aliens. This class has the following:
*             Properties : private int weight, height, char gender, string name;
*             Methods    : public Alien(int w, int h, char g, string n) --> the Constructor for this class
*                         : public int getWeight() --> getter function to get the weight of the Alien
*                         : public int getHeight() --> getter function to get the weight of the Alien
*                         : public char getGender() --> getter function to get the gender of the Alien
*                         : public string getAlienName() --> getter function to get the name of the Alien
*                         : public int getPrestige() --> getter function to get the prestige of the Alien
*                         : public int setGenderPoints() --> setter function to set the gender points of an Alien
*                         : public void setAlienName(string alienName) --> setter function to set the name of
                           the Alien
*             Overloaded Operators: '+' operator for "breeding"
*                                   '=' operator to compare two Aliens
*                                   '!=' operator to compare two Aliens
*                                   '>' operator to compare two Aliens
*                                   '>=' operator to compare two Aliens
*                                   '<' operator to compare two Aliens
*                                   '<=' operator to compare two Aliens
*
* Constructor : None

```

* Returns : N/A

* Author : Gilberto Gabon - Student No.: 270204759

*****/

class Alien

{

int weight, height;

char gender;

string name; // To store the name of the alien

public:

// Constructor

Alien(int w, int h, char g, string n)

{

weight = w;

height = h;

gender = g;

name = n;

}

// Getterfunctions

int getWeight()

{

return weight;

}

int getHeight()

{

return height;

}

int setGenderPoints()

{

if (toupper(gender) == 'M')

{

return 2;

}

else

{

return 3;

}

}

```
char getGender()
{
    return gender;
}

string getAlienName()
{
    return name;
}

// setter for alien name
void setAlienName(string alienName)
{
    name = alienName;
}

int getPrestige()
{
    return height * weight * setGenderPoints();
}

// Overload the '+' operator for "breeding"
Alien operator+(Alien const &theOtherAlien)
{
    srand(time(NULL)); // seed the randomizer

    Alien offSpringAlien = Alien(theOtherAlien.weight, theOtherAlien.height, theOtherAlien.gender,
theOtherAlien.name); // initialize the offSpringAlien with values from theOtherAlien

    offSpringAlien.weight = (this->weight + theOtherAlien.weight) / 2; // this->weight is the weight of
the Alien of the other 'operand' (left side), theOtherAlien.weight is the weight of the other Alien of the
other 'operand; (right side)
    offSpringAlien.height = (this->height + theOtherAlien.height) / 2; // this->height is the height of the
Alien of the other 'operand' (left side), theOtherAlien.height is the height of the other Alien of the other
'operand; (right side)
    offSpringAlien.gender = ((rand() % 2 + 1) == 1 ? 'M' : 'F');    // Randomize a number between 1 and
2 to represent Male or Female where Male = 1; Female = 2;
    offSpringAlien.name = " ";
    return offSpringAlien;
}

// Overload the '==' operator to compare two Aliens
bool operator==(Alien &obj)
{

```



```
    if (this->getPrestige() == obj.getPrestige())
    {
        return true;
    }
    else
    {
        return false;
    }
}

// Overload the '!=' operator to compare two Aliens
bool operator!=(Alien &obj)
{

    if (this->getPrestige() != obj.getPrestige())
    {
        return true;
    }
    else
    {
        return false;
    }
}

// Overload the '>' operator to compare two Aliens
bool operator>(Alien &obj)
{

    if (this->getPrestige() > obj.getPrestige())
    {
        return true;
    }
    else
    {
        return false;
    }
}

// Overload the '>=' operator to compare two Aliens
bool operator>=(Alien &obj)
{

    if (this->getPrestige() >= obj.getPrestige())
```

```
{
    return true;
}
else
{
    return false;
}
}

// Overload the '<' operator to compare two Aliens
bool operator<(Alien &obj)
{

    if (this->getPrestige() < obj.getPrestige())
    {
        return true;
    }
    else
    {
        return false;
    }
}

// Overload the '<=' operator to compare two Aliens
bool operator<=(Alien &obj)
{

    if (this->getPrestige() <= obj.getPrestige())
    {
        return true;
    }
    else
    {
        return false;
    }
}
};

// Displays Alien information
void displayAlien(Alien alien)
{
    cout << "\n-----" << endl;
    cout << "Alien Name: " << alien.getAlienName() << endl;
    cout << "  Weight: " << alien.getWeight() << endl;
```

```

    cout << "   Height: " << alien.getHeight() << endl;
    cout << "   Gender: " << alien.getGender() << " (gender point = " << alien.setGenderPoints() << ")" <<
endl;
    cout << "   Prestige: " << alien.getPrestige() << " (p = weight * height * genderPoints)" << endl;
    cout << "-----" << endl;
}

```

// Function to create the Aliens and save those Aliens in a vector variable to be returned to a calling function/routine

```

vector<Alien> createAliens()
{
    vector<Alien> Aliens; // variable to contain the created aliens

    // Create the aliens and save to the vector variable Aliens
    Aliens.push_back(Alien(3, 5, 'M', "Alien1")); // aka Alien1
    Aliens.push_back(Alien(4, 7, 'F', "Alien2")); // aka Alien2
    Aliens.push_back(Alien(6, 8, 'M', "Alien3")); // aka Alien3
    Aliens.push_back(Alien(7, 9, 'F', "Alien4")); // aka Alien4

    cout << "\nAlien Pairs Created" << endl;
    cout << "=====" << endl;

    for (int i = 0; i < (int)Aliens.size(); i++)
    {
        displayAlien(Aliens[i]);
    }

    return Aliens;
}

```

// This function calls createAliens() to create the Aliens and Offsprings are then created by using the overloaded operator '+'

```

vector<Alien> createOffsprings()
{
    vector<Alien> Aliens = createAliens(); // This creates the Aliens (Aliens1 to Aliens4) and displays the
aliens' information

    vector<Alien> AlienOffsprings; // Vectore variable of type 'Alien' to hold the created offsprings

    AlienOffsprings.push_back(Aliens[0] + Aliens[1]); // Offspring from Aliens[0] & Aliens[1]. The '+'
operator here has been overloaded
    AlienOffsprings.push_back(Aliens[2] + Aliens[3]); // Offspring from Aliens[2] & Aliens[3]. The '+'
operator here has been overloaded
}

```

```

    AlienOffsprings[0].setAlienName("Alien5");
    AlienOffsprings[1].setAlienName("Alien6");

    cout << "\nOffsprings created...Alien5 and Alien6";
    cout << "\n===== \n";

    for (int i = 0; i < (int)AlienOffsprings.size(); i++)
    {
        displayAlien(AlienOffsprings[i]);
    }

    return AlienOffsprings;
}

// This function calls createOffsprings(). createOffsprings() calls createAliens()
void comparePrestige()
{

    vector<Alien> AlienOffsprings = createOffsprings();

    Alien Alien5 = AlienOffsprings[0];
    Alien Alien6 = AlienOffsprings[1];

    cout << "\nOffspring Prestige Comparison" << endl;
    cout << "===== " << endl;
    if (Alien5 == Alien6)
    {
        cout << "Alien5 == Alien6 ? true" << endl;
    }
    else
    {
        cout << "Alien5 == Alien6 ? false" << endl;
    };

    if (Alien5 != Alien6)
    {
        cout << "Alien5 != Alien6 ? true" << endl;
    }
    else
    {
        cout << "Alien5 != Alien6 ? false" << endl;
    };

    if (Alien5 > Alien6)

```

```

{
    cout << "Alien5 > Alien6 ? true" << endl;
}
else
{
    cout << "Alien5 > Alien6 ? false" << endl;
};

if (Alien5 >= Alien6)
{
    cout << "Alien5 >= Alien6 ? true" << endl;
}
else
{
    cout << "Alien5 >= Alien6 ? false" << endl;
};

if (Alien5 < Alien6)
{
    cout << "Alien5 < Alien6 ? true" << endl;
}
else
{
    cout << "Alien5 < Alien6 ? false" << endl;
};

if (Alien5 <= Alien6)
{
    cout << "Alien5 <= Alien6 ? true" << endl;
}
else
{
    cout << "Alien5 <= Alien6 ? false" << endl;
};
}

```

```

/*****

```

```

* Title      :    CS-105 Development Principles-2 Assessment 1
* Function Name :    showMainMenu()
* Purpose    :    Function to show the Menu for the Administrator of the system.
* Parameters :    None
* Returns    :    None
* Author     :    Gilberto Gabon - Student No.: 270204759

```

```

*****/

```

```
void showMainMenu()
{

    int choice = 0;
    vector<string> menu = {
        "\n=====",
        "    Main Menu",
        "=====",
        "1. Create Alien Pairs    ",
        "2. Create Offsprings    ",
        "3. Compare Offspring Prestige",
        "4. Exit",
        "=====",
        ""};

    while (choice != 4)
    {
        choice = showMenu(menu);

        switch (choice)
        {
        case 1:
            createAliens();
            break;
        case 2:
            createOffsprings();
            break;
        case 3:
            comparePrestige();
            break;

        default:
            break;
        }
    }
}

int main()
{
    showMainMenu();
    return 0;
}
```

b. Process to solve the problem

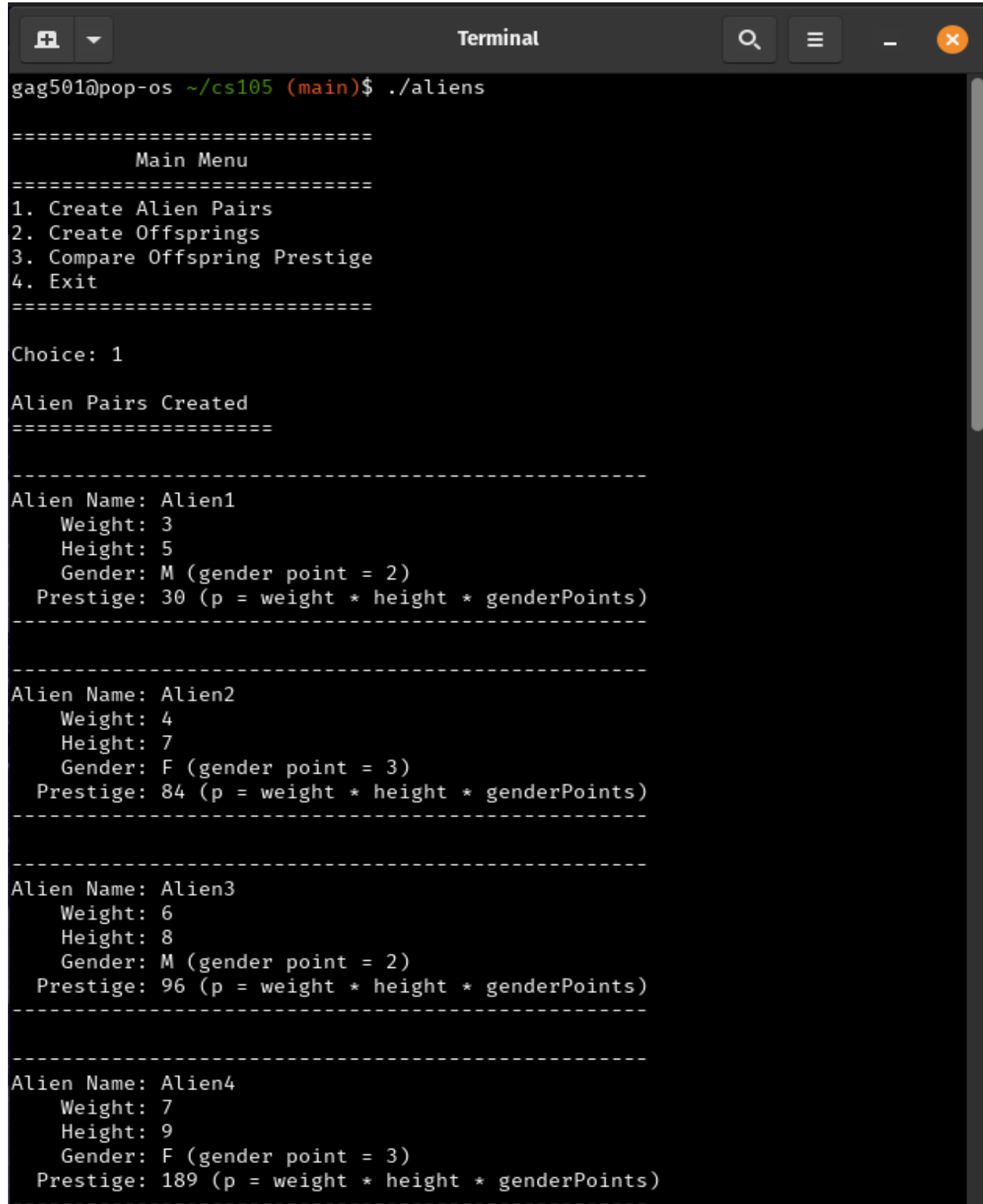
i. Class **Alien** was created. This class contains the following properties and methods:

- Properties : private int weight, height, char gender, string name;
 - Methods
 - a. public Alien(int w, int h, char g, string n) --> the Constructor for this class
 - b. public int getWeight() --> getter function to get the weight of the Alien
 - c. public int getHeight() --> getter function to get the weight of the Alien
 - d. public char getGender() --> getter function to get the gender of the Alien
 - e. public string getAlienName() --> getter function to get the name of the Alien
 - f. public int getPrestige() --> getter function to get the prestige of the Alien
 - g. public int setGenderPoints() --> setter function to set the gender points of an Alien
 - h. public void setAlienName(string alienName) --> setter function to set the name of the Alien
 - Overloaded Operators:
 - a. '+' operator for "breeding"
 - b. '==' operator to compare two Aliens
 - c. '!=' operator to compare two Aliens
 - d. '>' operator to compare two Aliens
 - e. '>=' operator to compare two Aliens
 - f. '<' operator to compare two Aliens
 - g. '<=' operator to compare two Aliens
- ii. Functions to display alien information - displayAlien(), create aliens – createAliens(), create offsprings – createOffsprings(), and compare prestige – comparePrestige(), show main menu – showMainMenu() were developed as support functions.
- iii. The displayAlien(Alien alien) function receives an object of type Alien. This function then displays the properties/information about the alien. Information is retrieved by calling the respective getter function, i.e., alien.getAlienName() to retrieve the name of the alien. Properties of the object alien have their corresponding getter functions.

- iv. Creating the aliens is done by calling the `createAliens()` function. This function returns a vector of type `Alien`. Inside this function, aliens are created with hard-coded properties, such the weight, height, gender, and name. This code can be further improved by asking these properties from the user. However, as this was not required in the brief, the author decided to create the aliens by hard coding their properties. There are four (4) aliens created, 2 males and 2 females. The function then loops around all the contents of the `Aliens` vector and display the property of each alien. The function then returns the vector of type `Alien` to the calling function for further use.
- v. To create the alien offsprings, the function `createOffsprings()` is called. This function returns a vector of type `alien`. Inside this function, the aliens are created first by calling the `createAliens()` function. The returned vector from calling the `createAliens()` function is saved in a vector variable (`Aliens`) of type `Alien`. Another vector variable of type `Alien` named `AlienOffsprings` is created. This variable will hold the offsprings that will be created from 'breeding'. The '+' operator has been overloaded to add two (2) aliens (male and female) and produce an offspring with properties as set out in the brief. Once all the offsprings are created, the function then displays the information of the alien offsprings by looping through the `AlienOffsprings` vector variable and calling the `displayAlien()` function to display each alien offspring information.
- vi. Comparing the prestige of the alien offsprings is done by calling the `comparePrestige()` function. This function does not return anything nor receives an argument. This function creates a vector variable `AlienOffsprings` through the call of the `createOffsprings()` function. Once the offsprings are created, this function then compares the two (2) springs (`Alien5` and `Alien6`) by using the overloaded operators ('==', '!=', '>', '>=', '<', '<='). The overloading of these operators are done inside the class definition of `Alien` class. Results of the comparisons are displayed to the user. Once done, this function returns to the calling function
- vii. Finally, the `main()` function is created. Inside the `main()` function, the `showMainMenu()` function is called. This function allows the user to choose the following menu:
 - Create Alien Pairs
 - Create Offsprings
 - Compare Offspring Prestige
 - Exit

Selecting the 'Create Alien Pairs' calls the `createAliens()` function, 'Create Offsprings' calls the `createOffsprings()` function, and 'Compare Offspring Prestige' calls the `comparePrestige()` function. Selecting 'Exit' ends the application.

c. Screenshot



```
gag501@pop-os ~/cs105 (main)$ ./aliens

=====
Main Menu
=====
1. Create Alien Pairs
2. Create Offsprings
3. Compare Offspring Prestige
4. Exit
=====

Choice: 1

Alien Pairs Created
=====

-----
Alien Name: Alien1
  Weight: 3
  Height: 5
  Gender: M (gender point = 2)
  Prestige: 30 (p = weight * height * genderPoints)
-----

-----
Alien Name: Alien2
  Weight: 4
  Height: 7
  Gender: F (gender point = 3)
  Prestige: 84 (p = weight * height * genderPoints)
-----

-----
Alien Name: Alien3
  Weight: 6
  Height: 8
  Gender: M (gender point = 2)
  Prestige: 96 (p = weight * height * genderPoints)
-----

-----
Alien Name: Alien4
  Weight: 7
  Height: 9
  Gender: F (gender point = 3)
  Prestige: 189 (p = weight * height * genderPoints)
-----
```

```
-----
Main Menu
-----
1. Create Alien Pairs
2. Create Offsprings
3. Compare Offspring Prestige
4. Exit
-----

Choice: 2

Alien Pairs Created
-----

Alien Name: Alien1
  Weight: 3
  Height: 5
  Gender: M (gender point = 2)
  Prestige: 30 (p = weight + height + genderPoints)
-----

Alien Name: Alien2
  Weight: 4
  Height: 7
  Gender: F (gender point = 3)
  Prestige: 84 (p = weight + height + genderPoints)
-----

Alien Name: Alien3
  Weight: 6
  Height: 8
  Gender: M (gender point = 2)
  Prestige: 96 (p = weight + height + genderPoints)
-----

Alien Name: Alien4
  Weight: 7
  Height: 9
  Gender: F (gender point = 3)
  Prestige: 189 (p = weight + height + genderPoints)
-----

Offsprings created...Alien5 and Alien6
-----

Alien Name: Alien5
  Weight: 3
  Height: 6
  Gender: M (gender point = 2)
  Prestige: 36 (p = weight + height + genderPoints)
-----

Alien Name: Alien6
  Weight: 6
  Height: 8
  Gender: M (gender point = 2)
  Prestige: 96 (p = weight + height + genderPoints)
-----
```

```
Terminal

-----
Choice: 3
Alien Pairs Created
-----

Alien Name: Alien1
  Weight: 3
  Height: 5
  Gender: M (gender point = 2)
  Prestige: 30 (p = weight * height * genderPoints)
-----

Alien Name: Alien2
  Weight: 4
  Height: 7
  Gender: F (gender point = 3)
  Prestige: 84 (p = weight * height * genderPoints)
-----

Alien Name: Alien3
  Weight: 6
  Height: 8
  Gender: M (gender point = 2)
  Prestige: 96 (p = weight * height * genderPoints)
-----

Alien Name: Alien4
  Weight: 7
  Height: 9
  Gender: F (gender point = 3)
  Prestige: 189 (p = weight * height * genderPoints)
-----

Offsprings created...Alien5 and Alien6
-----

Alien Name: Alien5
  Weight: 3
  Height: 6
  Gender: F (gender point = 3)
  Prestige: 54 (p = weight * height * genderPoints)
-----

Alien Name: Alien6
  Weight: 6
  Height: 8
  Gender: F (gender point = 3)
  Prestige: 144 (p = weight * height * genderPoints)
-----

Offspring Prestige Comparison
-----
Alien5 == Alien6 ? false
Alien5 != Alien6 ? true
Alien5 > Alien6 ? false
Alien5 >= Alien6 ? false
Alien5 < Alien6 ? true
Alien5 <= Alien6 ? true
```