

CS105 – Development Principles-2

# Assessment 2

Diploma in Software Development  
Yoobee Online

## CASE STUDY REPORT

**GILBERTO GABON**

Author

[270204759@yoobestudent.ac.nz](mailto:270204759@yoobestudent.ac.nz)

**1. Section 1 – Question 1: Pointer to Objects****a. Task 1: Code for the scenario – contains creation of classes and the output display:**

```

/*****
* Title      : CS-105 Development Principles-2 Assessment 2
* File       : healthapp.cpp
* Purpose    : Section 1 - Question 1: Pointer to Objects
*            : This program showcases the use of pointer to objects.
* Parameters : N/A
* Returns    : N/A
* Author     : Gilberto Gabon - Student No.: 270204759
*****/

#include <iostream>
#include <string>

using namespace std;

// define the class
class HealthActivity
{
private:
    int walkingSteps;
    float runningDistance;
    string name;

public:
    // Setter function to save user input into the object of the class HealthActivity
    void SetFunction(string nameUser, int stepsWalked, float kmsRun)
    {
        walkingSteps = stepsWalked;
        runningDistance = kmsRun;
        name = nameUser;
    }

    // Getter functions
    string GetName()
    {
        return name;
    }
    int GetSteps()
    {
        return walkingSteps;
    }
    float GetRuns()
    {
        return runningDistance;
    }
    // Display data from the class
    void GetFunction()

```

```

{
    cout << "Name: " << name << endl;
    cout << "Steps: " << walkingSteps << " steps" << endl;
    cout << "Walking + Running: " << runningDistance << " kms" << endl;
}
};

int main()
{
    HealthActivity users[5], *ptrUsers; // define an array of users and a pointer to the array of type HealthActivity class
    ptrUsers = users; // pointer to the array users of type HealthActivity
    string userName = "";
    int walkSteps = 0, steps = 0;
    float runKms = 0.00, runs = 0.00;
    float averageSteps = 0.00, averageDistance = 0.00;

    // get data inputs
    for (int i = 0; i < 5; i++)
    {
        cout << "Enter the name, number of steps and walking + running distance: ";
        cin >> userName;
        cin >> walkSteps;
        cin >> runKms;
        (ptrUsers + i)->SetFunction(userName, walkSteps, runKms); // save the inputted data into the array via the pointer by
        calling the SetFunction() method from the HealthActivity class
    }

    // display output
    for (int i = 0; i < 5; i++)
    {
        (ptrUsers + i)->GetFunction(); // Display the inputted data via a pointer by calling the GetFunction() method from
        the HealthActivity class
        steps += (ptrUsers + i)->GetSteps(); // sum the steps from the array of users via a pointer by calling the GetSteps()
        method from the HealthActivity class
        runs += (ptrUsers + i)->GetRuns(); // sum the distance from the array of users via a pointer by calling the GetRuns()
        method from the HealthActivity class
    }

    averageSteps = steps / 5; // Compute the average of 5 users
    averageDistance = runs / 5; // Compute the average of 5 users
    cout << "Average steps of 5 users: " << averageSteps << " steps" << endl;
    cout << "Average distance of walking + running for 5 users: " << averageDistance << " kms" << endl;

    return 0;
}

```

## b. Task 2: Process to solve the problem

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

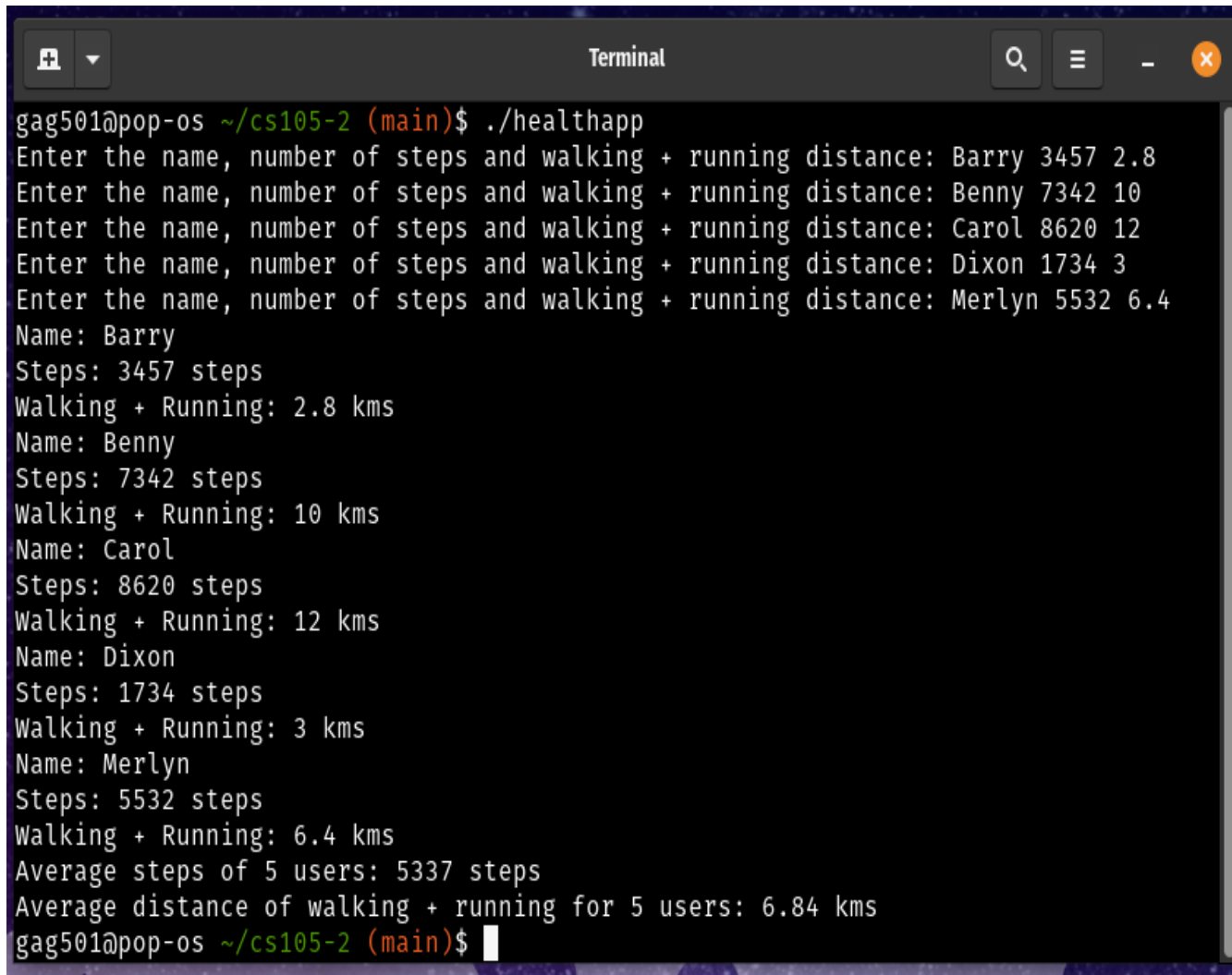
- Properties:

- Private: These are variables to store user information including number of steps walking and the distance travelled.
  1. **int walkingSteps;** //number of steps waked
  2. **float runningDistance;** // records distance covered
  3. **string name;** //name of the user
- Methods:
  - Public: These are getters and setter functions
    1. **void SetFunction(string nameUser, int stepsWalked, float kmsRun)** //Settter function to save user inputs into the object of the class HealthActivity.
    2. **string GetName()** //function to get the name
    3. **int GetSteps()** //function to get the number of steps taken
    4. **float GetRuns()** //function to get the distance run+waked
    5. **void GetFunction()** //function to display the user data (name, steps taken, and running+walking distance in kilometers).
  - ii. In the **main()** function, an array of objects of the class HealthActivity containing five (5) elements and a pointer of type HealthActivity class are defined. Variables to hold the user inputs, such as for the user name, the steps taken and the total distance in kilometers taken for walking and running are also defined.
  - iii. Next, a loop of five (5) iterations is created. Each iteration, user inputs for name, steps, and total distance for walking and running in kilometers are asked. Once all data is entered, the setter function SetFunction() is called via a pointer using pointer arithmetic:  
  
(ptrUsers + i)>SetFunction(userName, walkSteps, runKms) to save each input into the object of the class HealthActivity
  - iv. Once all the inputs the are done (five iterations), another loop is created to display the user inputs and to compute for the sum of all steps taken and the total distance travelled for walking and running. These

accumulated data are used later in the code to compute for the averages of steps and running and walking distances.

- v. Lastly, the average steps for the five (5) users and the average distance of walking and running for the five (5) are displayed.

### c. Screenshot



```

gag501@pop-os ~/cs105-2 (main)$ ./healthapp
Enter the name, number of steps and walking + running distance: Barry 3457 2.8
Enter the name, number of steps and walking + running distance: Benny 7342 10
Enter the name, number of steps and walking + running distance: Carol 8620 12
Enter the name, number of steps and walking + running distance: Dixon 1734 3
Enter the name, number of steps and walking + running distance: Merlyn 5532 6.4
Name: Barry
Steps: 3457 steps
Walking + Running: 2.8 kms
Name: Benny
Steps: 7342 steps
Walking + Running: 10 kms
Name: Carol
Steps: 8620 steps
Walking + Running: 12 kms
Name: Dixon
Steps: 1734 steps
Walking + Running: 3 kms
Name: Merlyn
Steps: 5532 steps
Walking + Running: 6.4 kms
Average steps of 5 users: 5337 steps
Average distance of walking + running for 5 users: 6.84 kms
gag501@pop-os ~/cs105-2 (main)$

```

## 2. Section 2 – Question 2: Polymorphism

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

```

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

```

```
* File      : videostore.cpp
* Purpose   : Section 2 - Question 2: Polymorphism
*           : This program showcases inheritance and polymorphism
* Parameters : N/A
* Returns    : N/A
* Author     : Gilberto Gabon - Student No.: 270204759
```

```
*****/

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

using namespace std;

// Base Class
class VideoGame
{
private:
    string title;
    float price;

public:
    // Virtual function to get data inputs from user. Made this virtual function for dynamic binding
    virtual void getVideoData()
    {
        cin.ignore(numeric_limits<std::streamsize>::max(), '\n'); // this clears the buffer thereby ensuring that any pending
        // newline character does not get fed into variable and eventually causing a skip in the input
        cout << "Enter title of game: ";
        getline(cin, title); // direct straight saving to the class' property
        cout << "    Enter price: ";
        cin >> price; // direct straight saving to the class' property
    }

    // Display data from the class. Make this function a virtual function to effect dynamic binding
    virtual void display()
    {
        cout << "\n*****" << endl;
        cout << "    Title: " << title << endl;
        cout << "    Price: " << price << endl;
    }
};

// Derived Class
class ComputerGame : public VideoGame
{
private:
    string os;

public:
    // display() function - function overriding - polymorphism. This display() function has added the display of the OS type.
    void display()
```

```

{
    VideoGame::display();           // call the display function of the base class
    cout << "    OS Type: " << os << endl; // then add the derived class' display method
    cout << "*****" << endl;
}

// getVideoData() function - function overriding - polymorphism. This getVideoData() function has added the input for
the OS of the game
void getVideoData()
{
    VideoGame::getVideoData();           // call the getVideoData() function of the base class
    cin.ignore(numeric_limits<std::streamsize>::max(), '\n'); // clear the buffer
    cout << "    Operating System: ";
    getline(cin, os); // save straight to the class property
}
};

// Derived Class
class ConsoleGame : public VideoGame
{
private:
    string os;

public:
    // display() function - function overriding - polymorphism. This display() function has added the display of the console
    type.
    void display()
    {
        VideoGame::display();           // call the display function of the base class
        cout << "Console Type: " << os << endl; // then add the derived class' display method
        cout << "*****" << endl;
    }

    // getVideoData() function - function overriding - polymorphism. This getVideoData() function has added the input for
    the console type of the game
    void getVideoData()
    {
        VideoGame::getVideoData();
        cin.ignore(numeric_limits<std::streamsize>::max(), '\n'); // clear the buffer
        cout << "    Console type: ";
        getline(cin, os); // direct - straight saving to the object's property
    }
};

int main()
{
    vector<VideoGame*> ptrVideoGames; // Define the vector array that will contain all the pointer to objects. This is a
    pointer variable that is a vector of type VideoGame

    ComputerGame *ptrgames; // Define the pointer to the game objects
    ConsoleGame *ptrconsoles; // Define the pointer to the console objects

```

```

// int choice = 1, ch = 1;
char choice = ' ', ch = ' ';

cout << "\n*****" << endl;
cout << "          Video Games Data Entry" << endl;
cout << "*****" << endl;

while (true)
{
    ptrgames = new ComputerGame; // always create a new object for every iteration
    ptrconsoles = new ConsoleGame; // always create a new object for every iteration

    cout << "Do you want to enter data for a Computer Game [o] or a Console Game [c]: ";

    cin >> ch;

    if (ch == 'o' || ch == 'O')
    {
        ptrgames->getVideoData(); // get games input - this made the code cleaner by placing the routine to take user
input inside the Base Class and the specific data inputs are then defined in the Derived classes.
        ptrVideoGames.push_back(ptrgames); // save into the array of pointers
    }
    else
    {
        if (ch == 'c' || ch == 'C')
        {
            ptrconsoles->getVideoData(); // get console inputs - this made the code cleaner by placing the routine to
take user input inside the Base Class and the specific data inputs are then defined in the Derived classes.
            ptrVideoGames.push_back(ptrconsoles); // save into the array of pointers
        }
        else
        {
            cout << "Please select only 'o' or 'c'" << endl;
            continue; // this causes to loop back to the top - avoiding the execution of code below
        }
    }

    cout << "\nDo you want to add another item (y/n)? ";
    cin >> choice;
    if (choice == 'n' || choice == 'N')
    {
        break;
    }
}

cout << "\n*****" << endl;
cout << "          Video Games List:" << endl;
cout << "*****" << endl;

for (int j = 0; j < (int)ptrVideoGames.size(); j++)
{
    ptrVideoGames[j]->display(); // polymorphism in action! The function display() here calls the relevant display()
function based on the kind of object contained in the current element of the array of pointers

```



```
}  
  
return 0;  
}
```

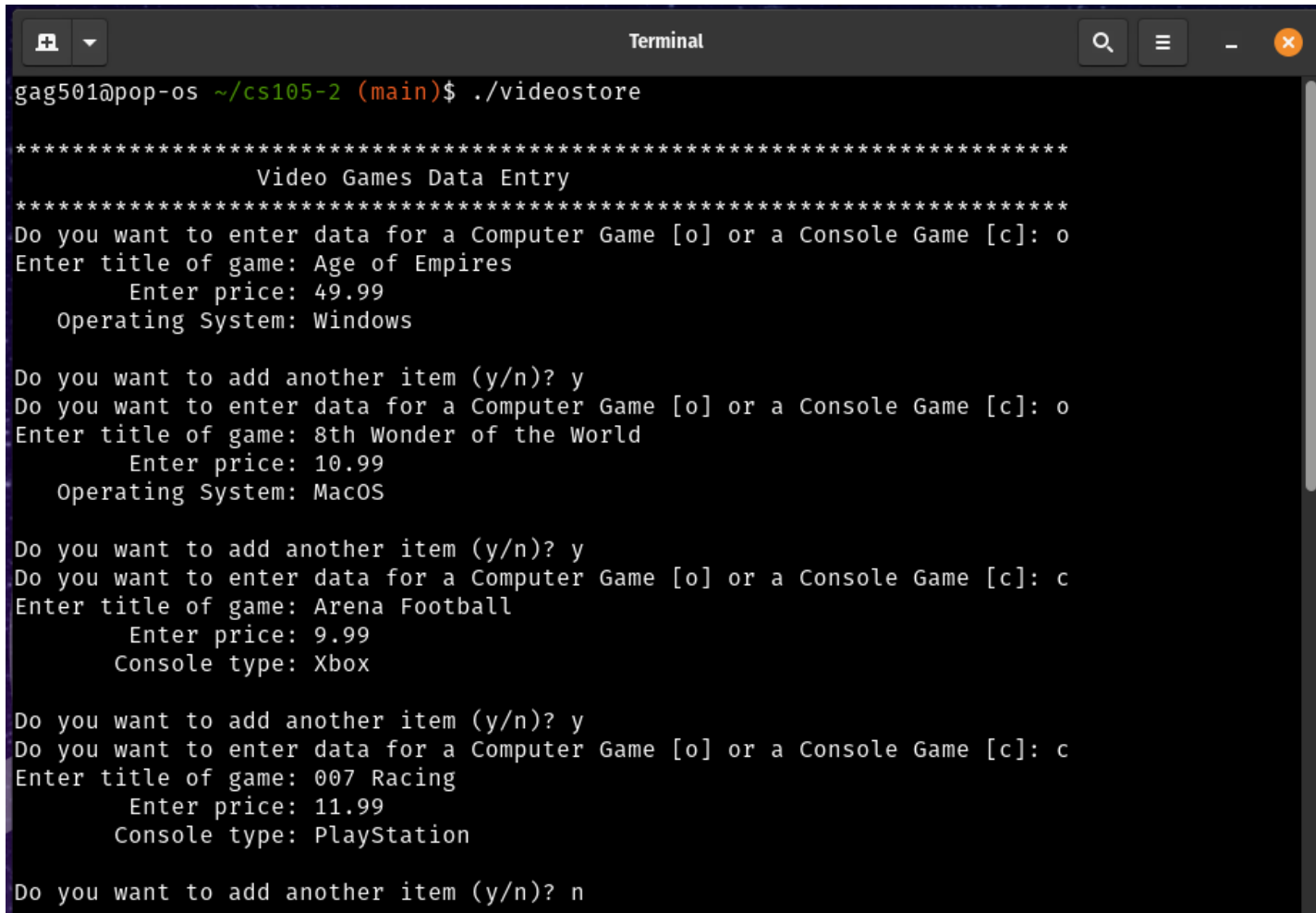
### b. Task 2: Process to solve the problem

- A class **VideoGame** was created first with the following:
  - Properties:
    - Private: These are variables to store information about a video
      1. string title; //holds the title of the game
      2. float price; //holds the price of the game
  - Methods:
    - Public: These are getters and setter functions
      1. **virtual void getVideoData()** // Virtual function to get data inputs from user. Made this virtual function for dynamic binding
      2. **virtual void display()** // Display data from the class. Made this function a virtual function to effect dynamic binding
- Next, two (2) Derived classes are created, namely ComputerGame and ConsoleGame. Each derived class has added its own properties and methods. Class ComputerGame and ConsoleGame, added a property 'os' to contain the operating system or the console type of each ComputerGame or ConsoleGame object. Added also in each of these derived classes are two (2) methods:
  1. **void display()** - which displays the video information inherited from the base class VideoGame plus the the added property 'os' that contains the operating system of the ComputerGame or the Console type object.
  2. **void getVideoData()** - this gets the user input for the title and price (inherited from the base class VideoGame) and the added property 'os' of the ComputerGame class or the console type of the ConsoleGame class.

- In the **main()** function, a vector of type VideoGame class pointer is defined - **ptrVideoGames**. Pointers for ComputerGame and ConsoleGame classes are also defined. Variables that will hold the user's menu choices are defined too ('choices', and 'ch').
- Following the variables definition area, the code goes into a 'while' loop to repeatedly ask the user for inputs (game title, price, and operating system/console type). Each iteration in the loop creates new objects for each of the derived classes. This loop continues until the user chooses to end it by answering the 'n' to the question whether to add another item or not.
- While inside the 'while' loop, the user is asked to choose between entering data for ComputerGame (o) or ConsoleGame (c). If the user chooses 'c', then program asks the user to enter the video data by calling the **ptrgames->getVideoData()** method. The call is made using the pointer **ptrgames**. Once all the data is entered, it is then added to the **vector array ptrVideoGame** to save the data. If the user chooses 'o', the program also asks for the user inputs by calling the **ptrconsoles->getVideoData()** method. The call is made using the pointer **ptrconsoles**. Once all the data is entered, the program saves the data into **vector array ptrVideoGames**.

The loop continues by asking the user if he/she wants to continue adding items. If the user answers yes, then the loop continues and repeats the process. If the user says no, then the program exits from the loop and displays the contents of the vector array pointer **ptrVideoGames**. The call to display is done inside a loop of finite length, i.e. the length of the vector array **ptrVideoGames**. Inside the loop, the **display()** method for each element in the vector array is called. Each element contains a pointer to an object of either from the derived class ComputerGame or ConsoleGame. By doing dynamic data binding, run-time polymorphism is effected.

## c. Screenshot



```
gag501@pop-os ~/cs105-2 (main)$ ./videostore

*****
                Video Games Data Entry
*****
Do you want to enter data for a Computer Game [o] or a Console Game [c]: o
Enter title of game: Age of Empires
    Enter price: 49.99
    Operating System: Windows

Do you want to add another item (y/n)? y
Do you want to enter data for a Computer Game [o] or a Console Game [c]: o
Enter title of game: 8th Wonder of the World
    Enter price: 10.99
    Operating System: MacOS

Do you want to add another item (y/n)? y
Do you want to enter data for a Computer Game [o] or a Console Game [c]: c
Enter title of game: Arena Football
    Enter price: 9.99
    Console type: Xbox

Do you want to add another item (y/n)? y
Do you want to enter data for a Computer Game [o] or a Console Game [c]: c
Enter title of game: 007 Racing
    Enter price: 11.99
    Console type: PlayStation

Do you want to add another item (y/n)? n
```

```

Terminal

*****
                        Video Games List:
*****

*****
    Title: Age of Empires
    Price: 49.99
    OS Type: Windows
*****

*****
    Title: 8th Wonder of the World
    Price: 10.99
    OS Type: MacOS
*****

*****
    Title: Arena Football
    Price: 9.99
    Console Type: Xbox
*****

*****
    Title: 007 Racing
    Price: 11.99
    Console Type: PlayStation
*****
gag501@pop-os ~/cs105-2 (main)$

```

### 3. Section 2 – Question 3: Operator Overloading

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

##### display

```

/*****
* Title    : CS-105 Development Principles-2 Assessment 2
* File     : complex.cpp
* Purpose  : Section 2 - Question 3: Operator Overloading
*          : This program showcases operator overloading for '+', '-', '*' and using this on complex numbers
* Parameters : N/A
* Returns  : N/A
* Author   : Gilberto Gabon - Student No.: 270204759
*****/

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

```

```
using namespace std;

class Complex
{
private:
    int a, b;

public:
    Complex(int var1, int var2)
    {
        a = var1;
        b = var2;
    }

    string GetComplexNum()
    {
        return to_string(a) + " + " + to_string(b) + "i";
    };

    // Overload the '+' operator to add two complex number and return the sum in a string format
    string operator+(Complex const &otherNum)
    {
        string retVal = "";
        int sumA = 0, sumB = 0;

        sumA = this->a + otherNum.a;
        sumB = this->b + otherNum.b;

        retVal = to_string(sumA) + " + " + to_string(sumB) + "i";

        return retVal;
    }

    // Overload the '-' operator to subtract two complex number and return the difference in a string format
    string operator-(Complex const &otherNum)
    {
        string retVal = "", optrString = " + ";
        int diffA = 0, diffB = 0;

        diffA = this->a - otherNum.a;
        diffB = this->b - otherNum.b;
        optrString = diffB < 0 ? " - " : " + ";

        retVal = to_string(abs(diffA)) + optrString + to_string(abs(diffB)) + "i";

        return retVal;
    }

    // Overload the '*' operator to multiply two complex number and return the result in a string format
    string operator*(Complex const &otherNum)
    {
        string retVal = "", optrString = " + ";
        int prodA = 0, prodB = 0;
```

```
prodA = (this->a * otherNum.a) - (this->b * otherNum.b);
prodB = (this->a * otherNum.b) + (this->b * otherNum.a);

retVal = to_string(prodA) + optrString + to_string(prodB) + "i";

return retVal;
}
};

void displayAddition(Complex num1, Complex num2)
{
    cout << "Addition" << endl;
    cout << "C1: " << num1.GetComplexNum() << endl;
    cout << "C2: " << num2.GetComplexNum() << endl;
    cout << "C3: " << num1 + num2 << endl; // add two complex numbers with the '+' operator as an overloaded operator
}

void displaySubtraction(Complex num1, Complex num2)
{
    cout << "Subtraction" << endl;
    cout << "C1: " << num1.GetComplexNum() << endl;
    cout << "C2: " << num2.GetComplexNum() << endl;
    cout << "C3: " << num1 - num2 << endl; // subtract two complex numbers with the '-' operator as an overloaded operator
}

void displayMultiplication(Complex num1, Complex num2)
{
    cout << "Multiplication" << endl;
    cout << "C1: " << num1.GetComplexNum() << endl;
    cout << "C2: " << num2.GetComplexNum() << endl;
    cout << "C3: " << num1 * num2 << endl; // multiply two complex numbers with the '*' operator as an overloaded operator
}

int main()
{
    Complex num1(3, 2); // initialize first complex number;
    int realNum, imaginaryNum;
    int choice = 0;

    cout << "1st Complex number: " << num1.GetComplexNum() << endl
         << endl;

    cout << "Enter 2nd Complex number values:" << endl;
    cout << "Enter real value: ";
    cin >> realNum;
    cout << "Enter imaginary value: ";
    cin >> imaginaryNum;

    Complex num2(realNum, imaginaryNum); // define complex number2
```

```

while (choice != 4)
{
    cout << "\nChoose Operation from Menu:" << endl;
    cout << "1. Addition" << endl;
    cout << "2. Subtraction" << endl;
    cout << "3. Multiplication" << endl;
    cout << "4. Exit" << endl;
    cout << "\nPlease enter your option: ";
    cin >> choice;
    switch (choice)
    {
    case 1:
        displayAddition(num1, num2);
        break;
    case 2:
        displaySubtraction(num1, num2);
        break;
    case 3:
        displayMultiplication(num1, num2);
        break;

    default:
        break;
    }
}

return 0;
}

```

## b. Task 2: Process to solve the problem

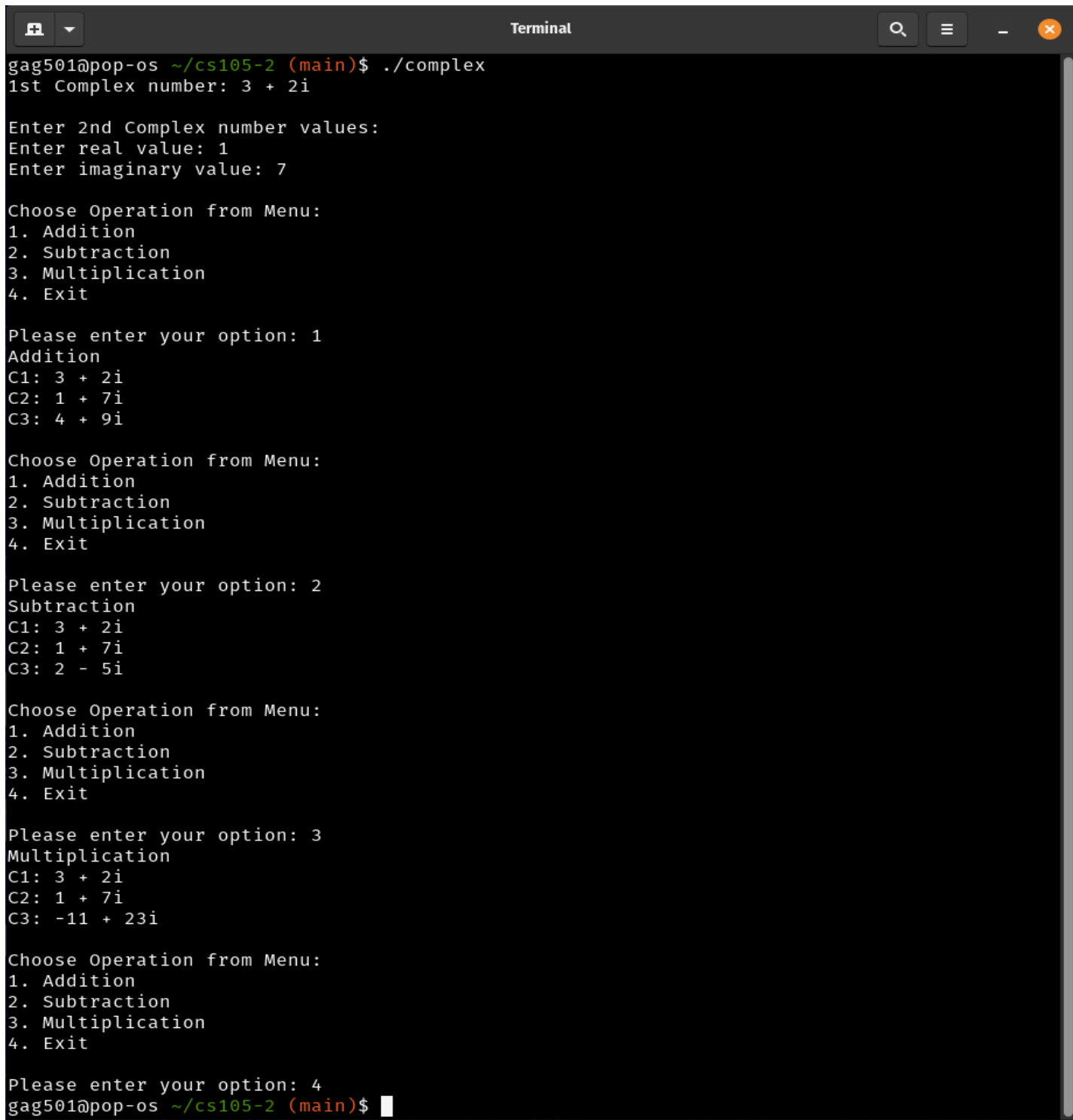
vi. A class **Complex** was created first with the following:

- Properties:
  - Private: These are variables to store information on the real and imaginary numbers.
    1. **int a;** //real number
    2. **Int b;** // imaginary number coefficient
- Methods:
  - Public: These are getters and overloading functions
    1. **Complex(int var1, int var2)** //Constructor function for the class
    2. **string GetComplexNum()** //returns a string for the complex number which contains the real number and the imaginary number
    3. **string operator+(Complex const &otherNum)** //overloading of the '+' operator to add two (2) complex numbers

4. **string operator-(Complex const &otherNum)**  
//overloading of the '-' operator to subtract two (2) complex numbers
  5. **string operator\*(Complex const &otherNum)**  
//overloading of the '\*' operator to multiply two (2) complex numbers
- **void displayAddition(Complex num1, Complex num2)** - utility function to display the result of the addition of two (2) complex numbers.
  - **void displaySubtraction(Complex num1, Complex num2)** - utility function to display the result of the subtraction of two (2) complex numbers.
  - **void displayMultiplication(Complex num1, Complex num2)** - utility function to display the result of the multiplication of two (2) complex numbers.
  - In the **main()** function, the first complex number is defined by creating a variable (object) named **num1** with the type Complex class. This object is created using the constructor of this class. Initial values are saved into this class through the constructor's execution.
  - The program then proceeds to ask for user inputs for the second complex numbers. Inputs asked from the user are the real and the coefficient of the imaginary number. The second complex number is created by calling the Complex class constructor and feeding into the constructor the real and imaginary numbers
  - Once done, the application continues and goes into a 'while' loop for the display of the application menu. The user is asked to choose from a range of different operations, i.e. Addition, Subtraction, Multiplication, Exit.
  - On selection of the operation, the appropriate function is called to display the result of operation of the two (2) complex numbers.



## b. Screenshot

A terminal window titled "Terminal" with a dark background and light text. The window shows the execution of a program named "complex". The user enters the command "gag501@pop-os ~/cs105-2 (main)\$ ./complex". The program prompts for the first complex number, which is "3 + 2i". It then prompts for the second complex number's real and imaginary values, which are "1" and "7" respectively. The program displays a menu with four options: 1. Addition, 2. Subtraction, 3. Multiplication, and 4. Exit. The user selects option 1. The program then displays the three complex numbers: C1: 3 + 2i, C2: 1 + 7i, and C3: 4 + 9i. It shows the menu again, and the user selects option 2. The program displays the three complex numbers: C1: 3 + 2i, C2: 1 + 7i, and C3: 2 - 5i. It shows the menu again, and the user selects option 3. The program displays the three complex numbers: C1: 3 + 2i, C2: 1 + 7i, and C3: -11 + 23i. It shows the menu again, and the user selects option 4. The program then displays the prompt "gag501@pop-os ~/cs105-2 (main)\$" with a cursor.

```
gag501@pop-os ~/cs105-2 (main)$ ./complex
1st Complex number: 3 + 2i

Enter 2nd Complex number values:
Enter real value: 1
Enter imaginary value: 7

Choose Operation from Menu:
1. Addition
2. Subtraction
3. Multiplication
4. Exit

Please enter your option: 1
Addition
C1: 3 + 2i
C2: 1 + 7i
C3: 4 + 9i

Choose Operation from Menu:
1. Addition
2. Subtraction
3. Multiplication
4. Exit

Please enter your option: 2
Subtraction
C1: 3 + 2i
C2: 1 + 7i
C3: 2 - 5i

Choose Operation from Menu:
1. Addition
2. Subtraction
3. Multiplication
4. Exit

Please enter your option: 3
Multiplication
C1: 3 + 2i
C2: 1 + 7i
C3: -11 + 23i

Choose Operation from Menu:
1. Addition
2. Subtraction
3. Multiplication
4. Exit

Please enter your option: 4
gag501@pop-os ~/cs105-2 (main)$
```

