

CS105 – Development Principles-2

Assessment 2

Diploma in Software Development
Yoobee Online

CASE STUDY REPORT

GILBERTO GABON

Author

270204759@yoobeestudent.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:
    HealthActivity(string username, int steps, float distance) // class constructor
    {
        name = username;
        walkingSteps = steps;
        runningDistance = distance;
    }
    // Getter functions
    string GetName()
    {
        return name;
    }
    int GetSteps()
    {
        return walkingSteps;
    }
};

```

```
    }
    float GetRuns()
    {
        return runningDistance;
    }
    // Display data from the class
    void displayData()
    {
        cout << "Name: " << name << endl;
        cout << "Steps: " << walkingSteps << " steps" << endl;
        cout << "Walking + Running: " << runningDistance << " kms" << endl;
    }
};
// this setFunction() receives an array of pointers to objects from the class HealthActivity
void setFunction(HealthActivity *ptrUsers[5])
{
    int walkSteps = 0;
    float runKms = 0.00;
    string name = "";
    for (int i = 0; i < 5; i++)
    {
        cout << "Enter the name, number of steps and walking + running distance: ";
        cin >> name;
        cin >> walkSteps;
        cin >> runKms;
        ptrUsers[i] = new HealthActivity(name, walkSteps, runKms); // create the object and save the
address of that object to the array
    }
}
// this getFunction() receives an array of pointers to objects from the class HealthActivity
void getFunction(HealthActivity *ptrUsers[5])
{
    int sumSteps = 0;
    float sumDistance = 0.00, avgSteps = 0.00, avgDistance = 0.00;
    for (int i = 0; i < 5; i++)
    {
        ptrUsers[i]->displayData();           // call the method to display the data of the current
object
        sumSteps += ptrUsers[i]->GetSteps();   // accumulate the number of steps taken
    }
}
```

```
        sumDistance += ptrUsers[i]->GetRuns(); // accumulate the distance taken
    }
    avgSteps = sumSteps / 5;           // computer for the average of the number of steps taken
    avgDistance = sumDistance / 5; // compute for the average of the distances taken
    cout << "Average steps of 5 users: " << avgSteps << " steps" << endl;
    cout << "Average distance of walking + running for 5 users: " << avgDistance << " kms" << endl;
}
int main()
{
    HealthActivity *ptrUsers[5]; // define an array of pointers to objects
    setFunction(ptrUsers);       // get user inputs
    getFunction(ptrUsers);       // display outputs
    return 0;
}
```

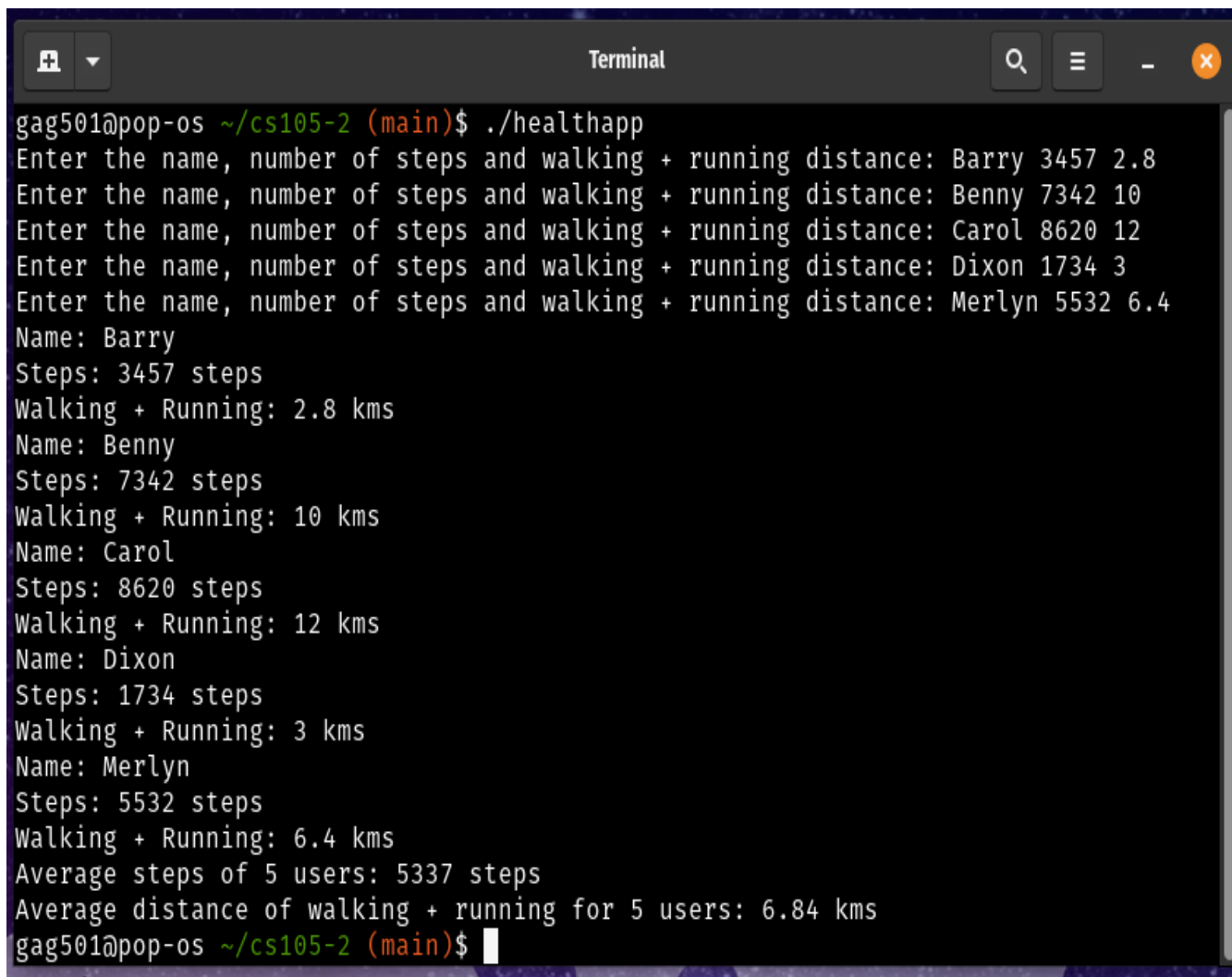
b. Task 2: Process to solve the problem

- 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 displayData()** //method to display contents of the object, i.e. name, steps, and walking+running distance
 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+walked
- ii. **void setFunction(HealthActivity *ptrUsers[5])** and **void getFunction(HealthActivity *ptrUsers[5])** were created next. Both of these functions receive an array of pointers to objects from the class **HealthActivity**.
- iii. The **setFunction()** is used to take inputs from the user. Inside a **for..loop**, the user is asked to enter the name, steps, and distance walked and run. Once inputs are done, a new object of the type **HealthActivity** is created. The address of this object is then saved into an array of pointers. The loop goes for five (5) users.
- iv. The **getFunction()** is used to display the contents of the array of pointers to objects of type **HealthActivity**. Inside this function, a call is made to the **display()** method of the object from the class **HealthActivity**. The total number of steps and the total distance covered are also computed inside

this function. The averages for the steps and distances covered are then computed and displayed.

- v. In the **main()** function, a pointer to an array of objects of the class HealthActivity containing five (5) elements is defined. Then the setFunction() is called with the pointer variable being passed as an argument. After the execution of the setFunction(), the getFunction() is then called with the pointer variable passed as an argument. This function displays the contents to the array of pointers.

c. Screenshot



```
Terminal
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
{
protected:
    string title;
    float price;
public:
    static int videoGameCount; // static data variable whose value is shared across all objects
    // Constructor of the base class
    VideoGame(string t, float p)
    {
        title = t;
        price = p;
    }
    // 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 operatingSystemType;
public:
    ComputerGame(string t, float p, string os) : VideoGame(t, p) // Constructor chaining
    {
        this->operatingSystemType = os;
    }
    // Setters and getters
    void setOs(string os)
    {
        this->operatingSystemType = os;
    }
    string getOs()
    {
        return this->operatingSystemType;
    }
    // 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: " << operatingSystemType << endl; // then add the derived class'
display method
        cout << "*****" << endl;
    }
};
// Derived Class
class ConsoleGame : public VideoGame
{

```



```

private:
    string consoleType;
public:
    ConsoleGame(string t, float p, string c) : VideoGame(t, p) // Constructor chaining
    {
        this->consoleType = c;
    }
    // Setters and getters
    void setConsole(string console)
    {
        this->consoleType = console;
    }
    string getConsole()
    {
        return this->consoleType;
    }
    // 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: " << consoleType << endl; // then add the derived class' display method
        cout << "*****" << endl;
    }
};
int VideoGame::videoGameCount = 0;
// Setter Function
void setComputerGame(VideoGame *gamePtr[5], int count)
{
    string title, osType;
    float price;
    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 << "Please enter title of computer game: ";
    getline(cin, title);
    cout << "Please enter price: ";
    cin >> price;
}

```

```
    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 << "Please enter operating system type: ";
    getline(cin, osType);
    gamePtr[count] = new ComputerGame(title, price, osType);
};
// Setter Function
void setConsoleGame(VideoGame *gamePtr[5], int count)
{
    string title, consoleType;
    float price;
    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 << "Please enter title of console game: ";
    getline(cin, title);
    cout << "Please enter price: ";
    cin >> price;
    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 << "Please enter console type: ";
    getline(cin, consoleType);
    gamePtr[count] = new ConsoleGame(title, price, consoleType);
};
int main()
{
    VideoGame *ptrVideoGames[5]; // Define the array that will contain all the pointer to objects.
    char choice = ' ', ch = ' ';
    cout << "\n*****" << endl;
    cout << "                Video Games Data Entry" << endl;
    cout << "*****" << endl;
    while (true)
    {
        cout << "\nDo you want to enter data for a Computer Game [o] or a Console Game [c]: ";
        cin >> ch;
        if (ch == 'o' || ch == 'O')
        {
```

```

        setComputerGame(ptrVideoGames, VideoGame::videoGameCount);
        VideoGame::videoGameCount++;
    }
    else
    {
        if (ch == 'c' || ch == 'C')
        {
            setConsoleGame(ptrVideoGames, VideoGame::videoGameCount);
            VideoGame::videoGameCount++;
        }
        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' || VideoGame::videoGameCount > 4)
    {
        break;
    }
};
cout << "\n*****" << endl;
cout << "                        Video Games List:" << endl;
cout << "*****" << endl;
for (int j = 0; j < VideoGame::videoGameCount; 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:
 - Protected: 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
 - Public:
 1. **static int videoGameCount;** //this property will hold the count of videogame objects created so far.
 - Methods:
 - Public: These are getters and setter functions
 1. **VideoGame(string t, float p)** // Constructor function for the base class
 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**. The constructor function for each of these derived classes utilises what is called “constructor chaining”, where this calls the base class constructor and added its own unique properties for the class. Each derived class has added its own property and methods. Class **ComputerGame** and **ConsoleGame**, added their respective properties ‘operatingSystemType’ and ‘consoleType’ to contain the operating system and the console type. Added also in each of these derived classes are the following methods:
 1. **ComputerGame** class:
 - a. **void setOs(string os)** //setter function to set the added property ‘operatingSystemType’
 - b. **string getOs()** //getter function to retrieve value for the property ‘operatingSystemType’.

- c. **void display()** // displays the video information inherited from the base class VideoGame plus the added property 'operatingSystemType' that contains the operating system of the ComputerGame.

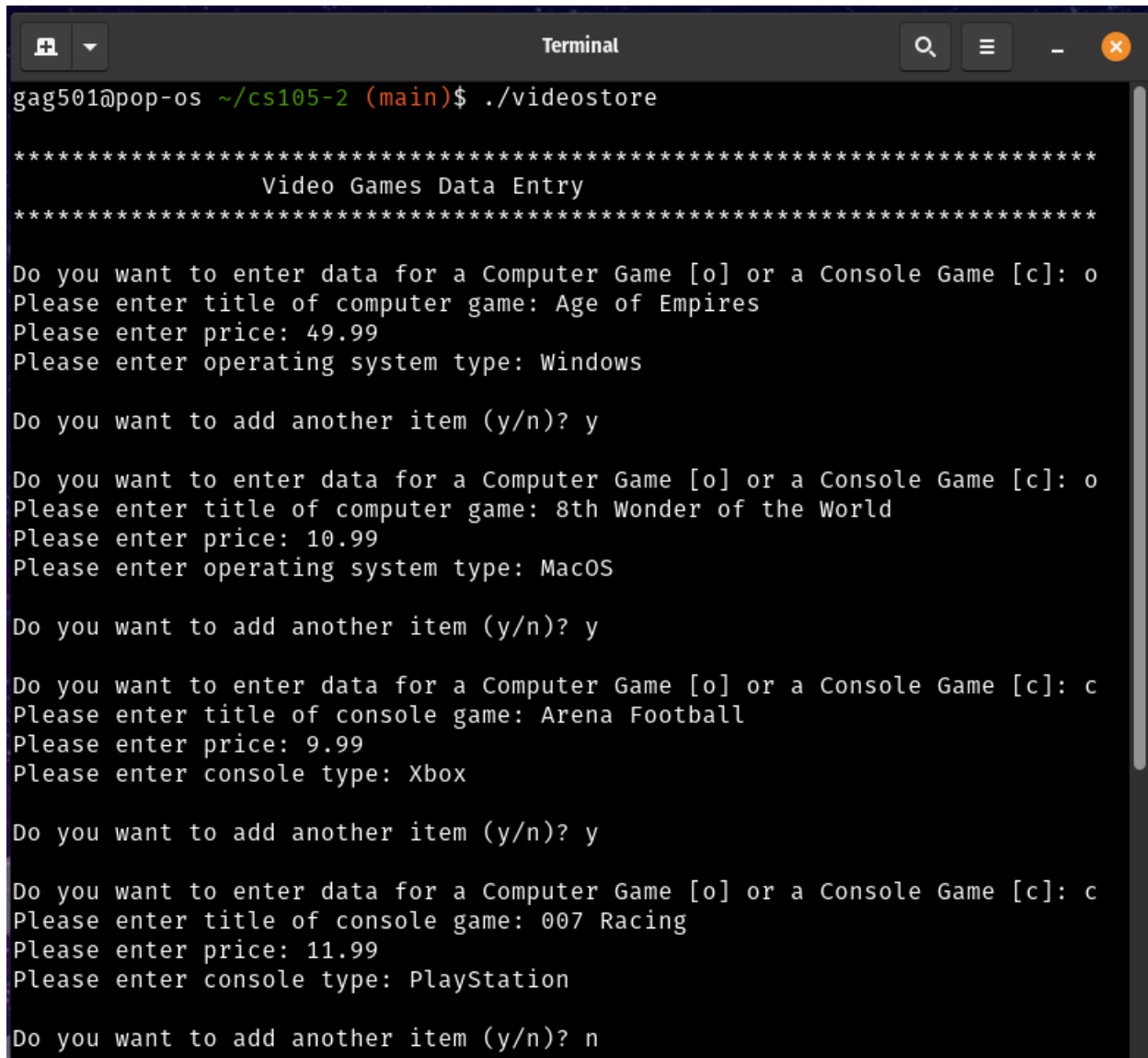
2. **ConsoleGame** class:

- a. **void setConsole(string console)**//setter function to set the added property 'consoleType '
- b. **string getOs()** //getter function to retrieve value for the property 'consoleType '.
- c. **void display()** // displays the video information inherited from the base class VideoGame plus the added property 'consoleType ' that contains the console type of the ConsoleGame.

- Next in the line is the definition of setter functions for the ComputerGame and ConsoleGame classes: **void setComputerGame(VideoGame *gamePtr[5], int count)** and **void setConsoleGame(VideoGame *gamePtr[5], int count)**, respectively. Both of these functions receive an array of pointers of type VideoGame and the current count of the VideoGame object. Inside both of these functions are codes to ask for user's input for the title, price, and operating system or console type (depending on the object being entered). Each object that is created is saved into the *gamePtr array of pointers.
- In the **main()** function, an array of five (5) elements that will contain the pointer to the videogame objects is defined. Also defined are variables to contain the choices the user makes when asked for the type of entry and if the user wants to continue adding a new videogame object.
- Following the variables definition area, the code goes into a 'while' loop. Inside the while loop, the system asks the user what type of entry he/she is making. If the user chooses 'o', then the program asks the user to enter the video data for the computer game by calling the setter function

setComputerGame(ptrVideoGames, VideoGame::videoGameCount); The user then enters data for the computer game object. After the execution of this setter function, the **VideoGame::videoGameCount** variable is incremented. The same process goes to the execution of the setter function **setConsoleGame(ptrVideoGames, VideoGame::videoGameCount)** if the user chooses, 'c'. After this execution of the code, the system then increments **VideoGame::videoGameCount** variable to track down the number of objects created so far. 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 (max of 5 iterations). If the user says no, then the program exits from the loop and displays the contents of the array pointer **ptrVideoGames**. The call to display is done inside a loop of maximum of five (5) iteration. Inside the loop, the display() method for each element in the 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

A terminal window titled "Terminal" with standard macOS window controls (red, yellow, green buttons) in the top-left corner. The terminal shows a user at the prompt "gag501@pop-os ~/cs105-2 (main)\$" running the command "./videostore". The program outputs a title screen "***** Video Games Data Entry *****". It then prompts the user to enter data for a Computer Game [o] or a Console Game [c]. The user enters 'o'. The program asks for the title, price, and operating system type. The user enters "Age of Empires", "49.99", and "Windows". It then asks if the user wants to add another item (y/n), and the user enters 'y'. This process repeats for two more items: "8th Wonder of the World" (price 10.99, MacOS) and "Arena Football" (price 9.99, Xbox). Finally, the user enters 'c' for a console game, "007 Racing" (price 11.99, PlayStation), and then enters 'n' when asked if they want to add another item.

```
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
Please enter title of computer game: Age of Empires
Please enter price: 49.99
Please enter operating system type: 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
Please enter title of computer game: 8th Wonder of the World
Please enter price: 10.99
Please enter operating system type: 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
Please enter title of console game: Arena Football
Please enter price: 9.99
Please enter 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
Please enter title of console game: 007 Racing
Please enter price: 11.99
Please enter console type: PlayStation

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

```
Terminal

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
Please enter title of console game: 007 Racing
Please enter price: 11.99
Please enter console type: PlayStation

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

*****
                        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;
}
```

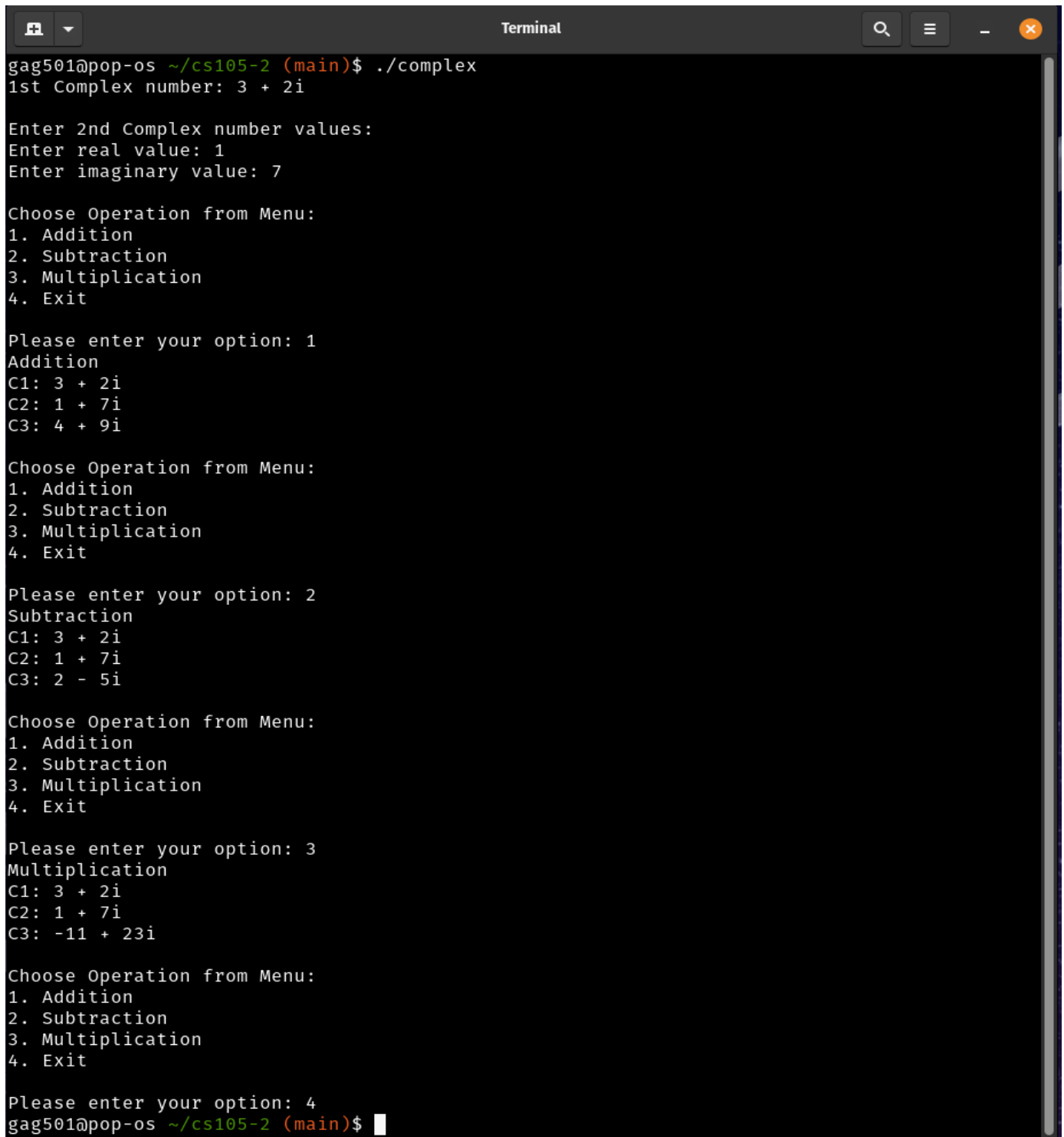
b. Task 2: Process to solve the problem

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

```
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)$
```