San Francisco Bay University

CS360L - Programming in C and C++ Lab
Lab Assignment #3

Due day: 3/8/2024

Instruction:

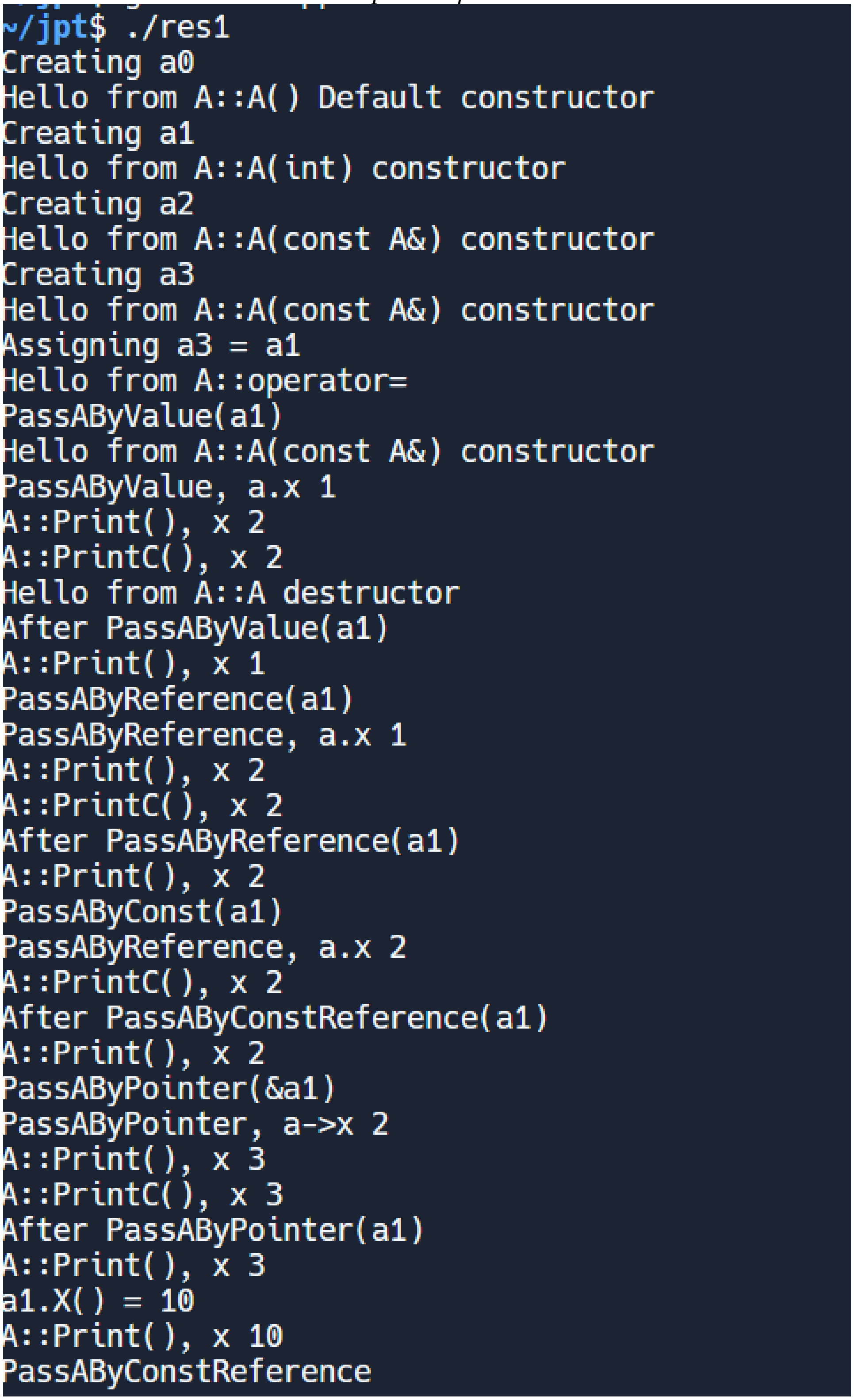
1. Push the answer sheets/source code to Github
2. Please follow the code style rule like programs on handout.
3. Overdue lab assignment submission can't be accepted.
4. Take academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)

1. Analyze the following program and explain each statement and commented-down statements in red. Finally, run the program and type in appropriate inputs from standard input device to show the running results

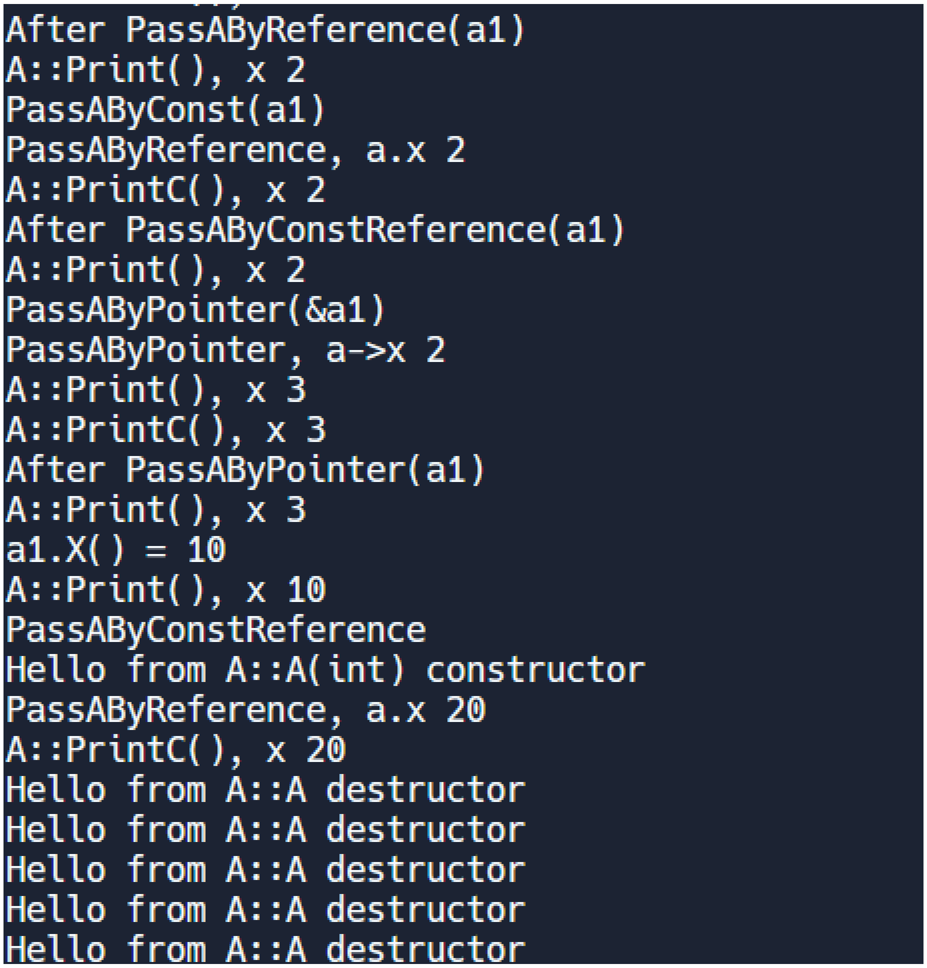
```
#include <stdio.h>     → stdio.h is a header file which
has prototypes and imp functions
#include <iostream>    → iostream is a input output
handling file

using namespace std;           → uses the standard
namespace

class A {          -> create a class
public:
  A();          // →default
  A(int);       // → integer parameter
  A(const A&); //→ constant reference
                //
  ~A();         // → deconstructor
public:
  void operator=(const A& rhs); // function1
  void Print();         //           f2
  void PrintC() const; //           f3
                        //
  int x;        //                var x
public:
  //
  int& X() { return x; }          f4
};
```

```
A::A()
  : x(0)              → initializing the variable x to 0
{
  cout << "Hello from A::A() Default constructor" << endl;
}

A::A(int i)       → initializing the variable x to i
    : x(i)
{
  cout << "Hello from A::A(int) constructor" << endl;
}

A::A(const A& a)  → initializing the new objects value as
the copy of other object of the same class A
    : x(a.x)
{
  cout << "Hello from A::A(const A&) constructor" << endl;
}

A::~A()  →deconstructs , when members out of scope
{
  cout << "Hello from A::A destructor" << endl;
}

void A::operator=(const A& rhs)→ copies a content from a
object to another
{
  x = rhs.x;
  cout << "Hello from A::operator=" << endl;
}

void A::Print()  → printing the value of x
{
  cout << "A::Print(), x " << x << endl; printing the
value of x (constant )
}

void A::PrintC() const   → print function for constant
member or function
{
  cout << "A::PrintC(), x " << x << endl;
}

void PassAByValue(A a)  function taking an object of class
A by value
{
  cout << "PassAByValue, a.x " << a.x << endl; print x
value
  a.x++;  // increment value of x
  a.Print(); print function
  a.PrintC(); print function for constants
}

void PassAByReference(A& a) function taking nan object of
class A by reference
{
  cout << "PassAByReference, a.x " << a.x << endl; print
value x
  a.x++;  // increment value of x
  a.Print(); print function
  a.PrintC();print constant function
}

void PassAByConstReference(const A& a) function taking
object of class A by const reference
{
  cout << "PassAByReference, a.x " << a.x << endl;
  a. PrintC();

  //a.Print(); // Call to "non-const" print function fails!
  // Compiler error from above line.  Why?
}

void PassAByPointer(A* a) function taking a pointet to an
object of class A
{
  cout << "PassAByPointer, a->x " << a->x << endl;
  a->x++; printing value of x using pointer notation
  a->Print();
  a->PrintC();
}


int main()
{
  cout << "Creating a0"; getchar();
  A a0;    //  creating object of class A

  cout << "Creating a1"; getchar();
  A a1(1); // creating object with parametrized
constructor

  cout << "Creating a2"; getchar();
  A a2(a0); // creating object by copying from another
object

  cout << "Creating a3"; getchar();
  A a3 = a0; //  creating object by copying from another
object

  cout << "Assigning a3 = a1"; getchar();
  a3 = a1; // assigning one object to another using the
assignment operator

  // Call some of the "A" subroutines
  cout << "PassAByValue(a1)"; getchar();
  PassAByValue(a1); // calling a function with an object
passed by value
  cout << "After PassAByValue(a1)" << endl; function call
  a1.Print();

  cout << "PassAByReference(a1)"; getchar();
  PassAByReference(a1); // calling a function with an
object passed by reference
  cout << "After PassAByReference(a1)" << endl;
  a1.Print();

  cout << "PassAByConst(a1)"; getchar();
  PassAByConstReference(a1); // calling a function with an
object passed by const reference
  cout << "After PassAByConstReference(a1)" << endl;
  a1.Print();

  cout << "PassAByPointer(&a1)"; getchar();
  PassAByPointer(&a1); // calling a function with a
pointer to an object
  cout << "After PassAByPointer(a1)" << endl;
  a1.Print();

  //
  cout << "a1.X() = 10"; getchar();
  a1.X() = 10; assigning a new value to x using member
function
  a1.Print();

  cout << "PassAByConstReference"; getchar();
  PassAByConstReference(20); calling a function with a
constant value

  // Why does the above compile?  What does it do?
  return 0;
}
```

➔ *The above compiles as follows :*



```
~/jpt$ ./res1
Creating a0
Hello from A::A() Default constructor
Creating a1
Hello from A::A(int) constructor
Creating a2
Hello from A::A(const A&) constructor
Creating a3
Hello from A::A(const A&) constructor
Assigning a3 = a1
Hello from A::operator=
PassAByValue(a1)
Hello from A::A(const A&) constructor
PassAByValue, a.x 1
A::Print(), x 2
A::PrintC(), x 2
Hello from A::A destructor
After PassAByValue(a1)
A::Print(), x 1
PassAByReference(a1)
PassAByReference, a.x 1
A::Print(), x 2
A::PrintC(), x 2
After PassAByReference(a1)
A::Print(), x 2
PassAByConst(a1)
PassAByReference, a.x 2
A::PrintC(), x 2
After PassAByConstReference(a1)
A::Print(), x 2
PassAByPointer(&a1)
PassAByPointer, a->x 2
A::Print(), x 3
A::PrintC(), x 3
After PassAByPointer(a1)
A::Print(), x 3
a1.X() = 10
A::Print(), x 10
PassAByConstReference
```

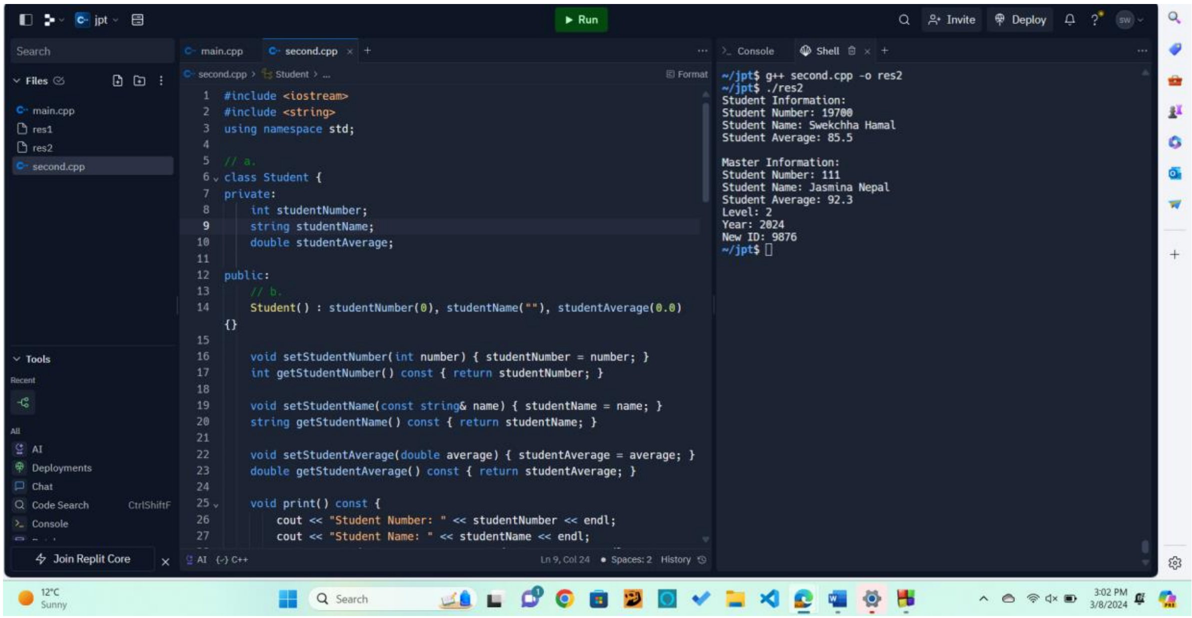


```
After PassAByReference(a1)
A::Print(), x 2
PassAByConst(a1)
PassAByReference, a.x 2
A::PrintC(), x 2
After PassAByConstReference(a1)
A::Print(), x 2
PassAByPointer(&a1)
PassAByPointer, a->x 2
A::Print(), x 3
A::PrintC(), x 3
After PassAByPointer(a1)
A::Print(), x 3
a1.X() = 10
A::Print(), x 10
PassAByConstReference
Hello from A::A(int) constructor
PassAByReference, a.x 20
A::PrintC(), x 20
Hello from A::A destructor
Hello from A::A destructor
Hello from A::A destructor
Hello from A::A destructor
Hello from A::A destructor
```

Class A objects (a0, a1, a2, a3) are created , and values are assigned to them. Using PassAByValue, PassAByReference, PassAByConstReference, and PassAByPointer, among other methods, it illustrates the creation, assignment, and manipulation of objects. The values of the objects provided to these functions are altered or printed. Lastly, when an item leaves its scope, the program records that it was destroyed using deconstructor.

2. Write the program based on the following requirements

   a. Define a class called *student* that has the following data members:
      i. *int* student number
      ii. *string* student name
      iii. *double* student average

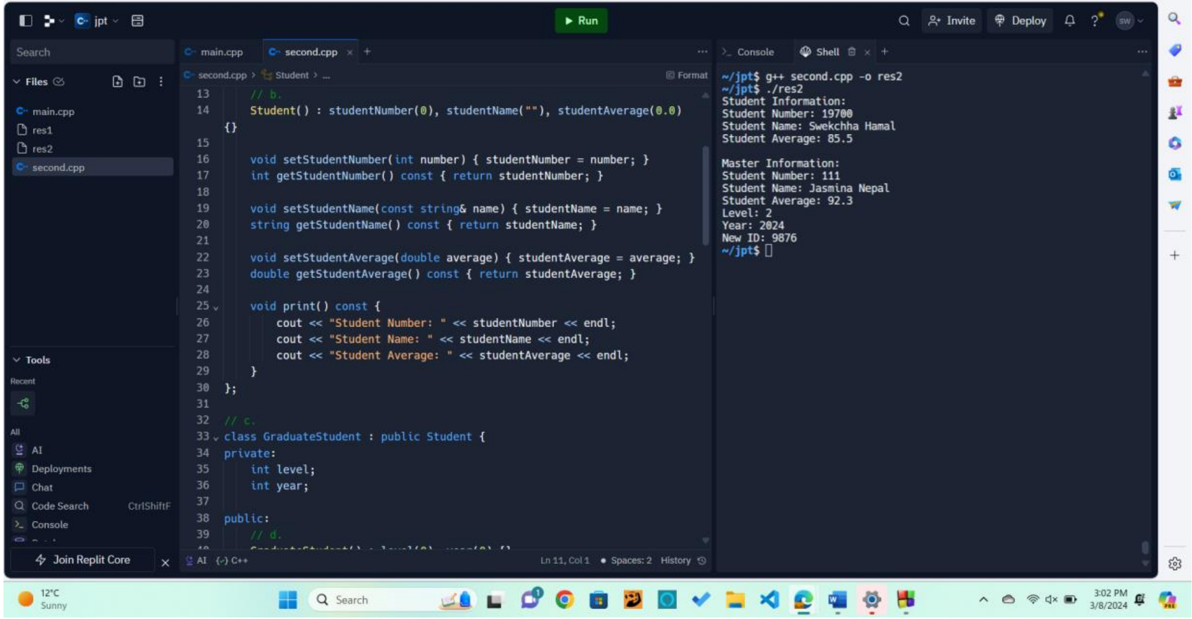


```cpp
#include <iostream>
#include <string>
using namespace std;

// a.
class Student {
private:
    int studentNumber;
    string studentName;
    double studentAverage;

public:
    // b.
    Student() : studentNumber(0), studentName(""), studentAverage(0.0)
{}

    void setStudentNumber(int number) { studentNumber = number; }
    int getStudentNumber() const { return studentNumber; }

    void setStudentName(const string& name) { studentName = name; }
    string getStudentName() const { return studentName; }

    void setStudentAverage(double average) { studentAverage = average; }
    double getStudentAverage() const { return studentAverage; }

    void print() const {
        cout << "Student Number: " << studentNumber << endl;
        cout << "Student Name: " << studentName << endl;
```

```
~/jpt$ g++ second.cpp -o res2
~/jpt$ ./res2
Student Information:
Student Number: 19700
Student Name: Swekchha Hamal
Student Average: 85.5

Master Information:
Student Number: 111
Student Name: Jasmina Nepal
Student Average: 92.3
Level: 2
Year: 2024
New ID: 9876
~/jpt$ ▯
```

b. The following member functions:

   i. Constructor that initialize the data members with default values.
   ii. *set* and *get* functions for each data member
   iii. *Print* function to print the values of data members.

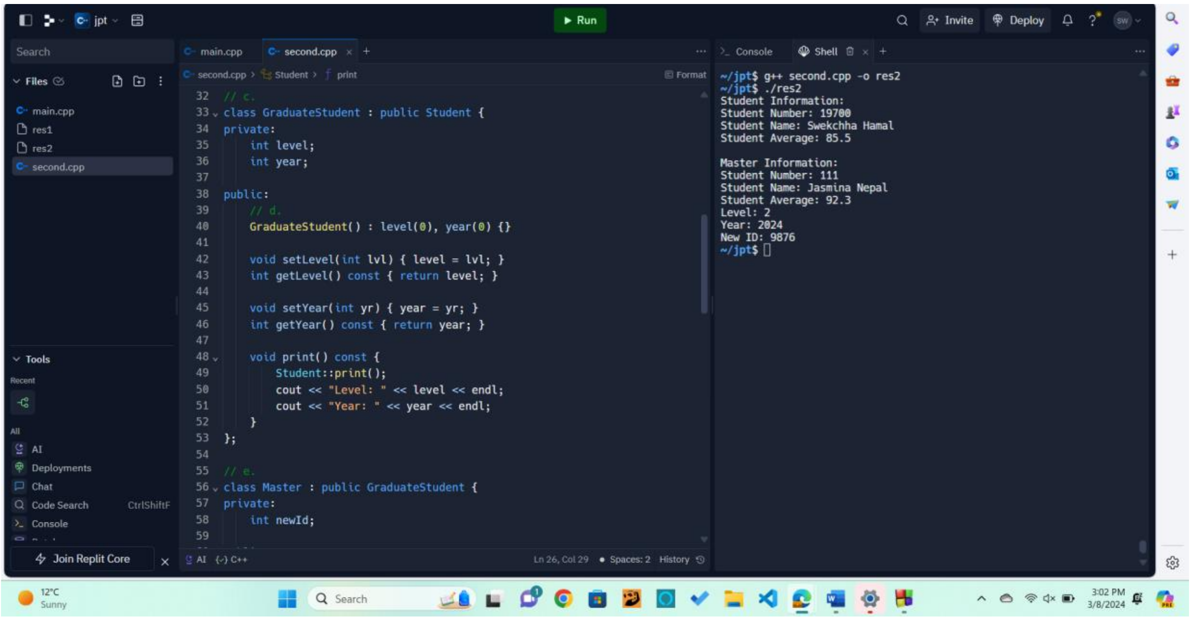


```cpp
    // b.
    Student() : studentNumber(0), studentName(""), studentAverage(0.0)
{}

    void setStudentNumber(int number) { studentNumber = number; }
    int getStudentNumber() const { return studentNumber; }

    void setStudentName(const string& name) { studentName = name; }
    string getStudentName() const { return studentName; }

    void setStudentAverage(double average) { studentAverage = average; }
    double getStudentAverage() const { return studentAverage; }

    void print() const {
        cout << "Student Number: " << studentNumber << endl;
        cout << "Student Name: " << studentName << endl;
        cout << "Student Average: " << studentAverage << endl;
    }
};

// c.
class GraduateStudent : public Student {
private:
    int level;
    int year;

public:
    // d.
```

```
~/jpt$ g++ second.cpp -o res2
~/jpt$ ./res2
Student Information:
Student Number: 19700
Student Name: Swekchha Hamal
Student Average: 85.5

Master Information:
Student Number: 111
Student Name: Jasmina Nepal
Student Average: 92.3
Level: 2
Year: 2024
New ID: 9876
~/jpt$ ▯
```

c. Define a class called *graduatestudent* that inherits data members and functions from the class *student*, and then declare the following data members :

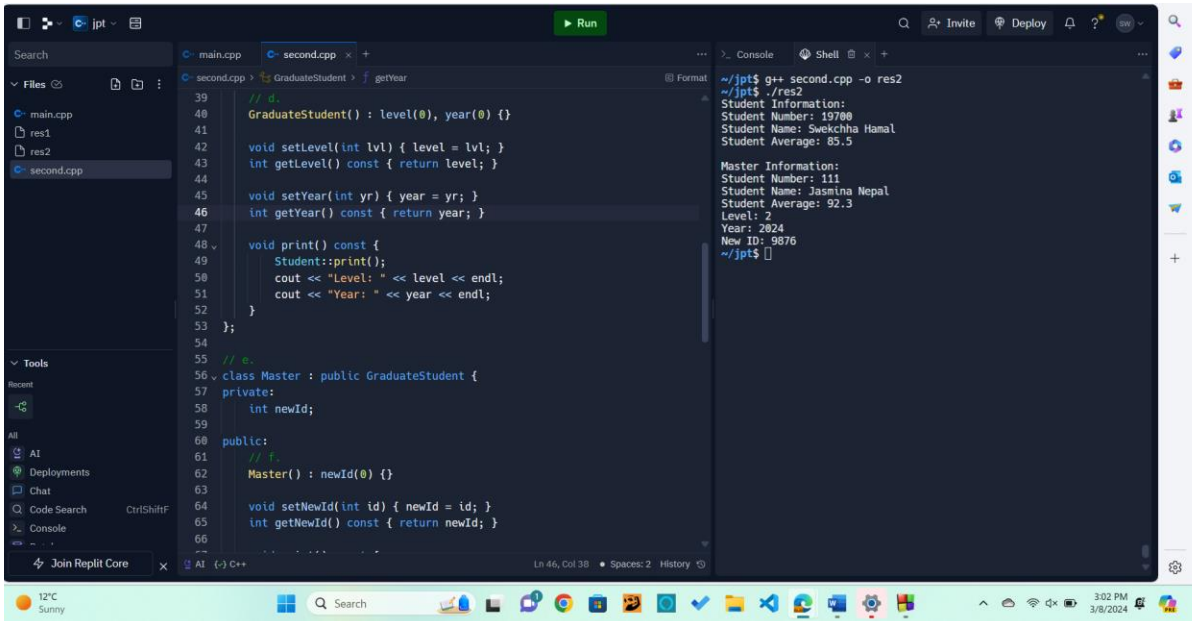
   i. int level
   ii. int year



```cpp
// c.
class GraduateStudent : public Student {
private:
    int level;
    int year;

public:
    // d.
    GraduateStudent() : level(0), year(0) {}

    void setLevel(int lvl) { level = lvl; }
    int getLevel() const { return level; }

    void setYear(int yr) { year = yr; }
    int getYear() const { return year; }

    void print() const {
        Student::print();
        cout << "Level: " << level << endl;
        cout << "Year: " << year << endl;
    }
};

// e.
class Master : public GraduateStudent {
private:
    int newId;

```

```
~/jpt$ g++ second.cpp -o res2
~/jpt$ ./res2
Student Information:
Student Number: 19700
Student Name: Swekchha Hamal
Student Average: 85.5

Master Information:
Student Number: 111
Student Name: Jasmina Nepal
Student Average: 92.3
Level: 2
Year: 2024
New ID: 9876
~/jpt$
```

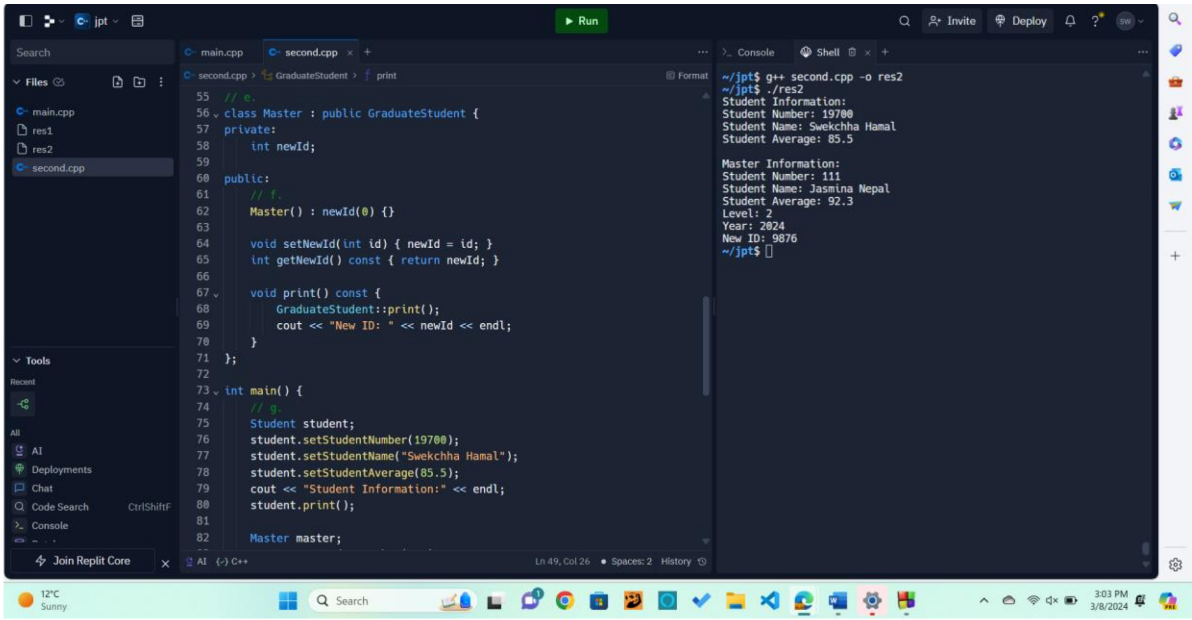
d. Member functions:
   i. constructor
   ii. *set* and *get* functions for each data member
   iii. Print function.



```cpp
    // d.
    GraduateStudent() : level(0), year(0) {}

    void setLevel(int lvl) { level = lvl; }
    int getLevel() const { return level; }

    void setYear(int yr) { year = yr; }
    int getYear() const { return year; }

    void print() const {
        Student::print();
        cout << "Level: " << level << endl;
        cout << "Year: " << year << endl;
    }
};

// e.
class Master : public GraduateStudent {
private:
    int newId;

public:
    // f.
    Master() : newId(0) {}

    void setNewId(int id) { newId = id; }
    int getNewId() const { return newId; }

```

```
~/jpt$ g++ second.cpp -o res2
~/jpt$ ./res2
Student Information:
Student Number: 19700
Student Name: Swekchha Hamal
Student Average: 85.5

Master Information:
Student Number: 111
Student Name: Jasmina Nepal
Student Average: 92.3
Level: 2
Year: 2024
New ID: 9876
~/jpt$
```

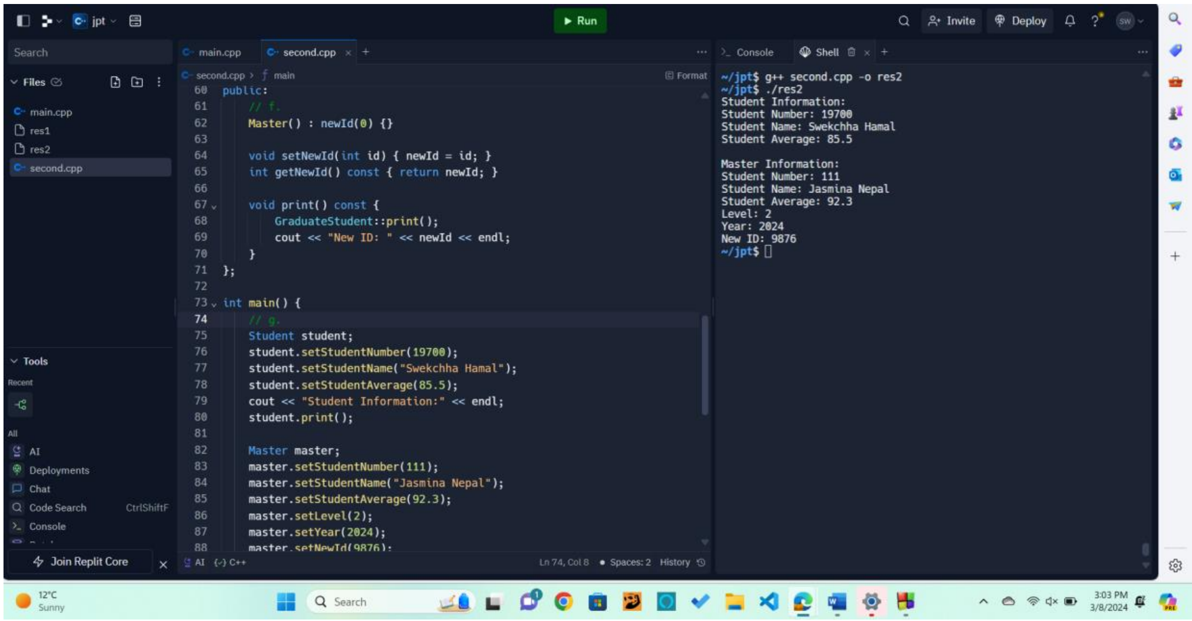
e. Define a class called *master* that inherits data members and functions from *graduatestudent* class, and then declare the following data member:
   i. int newid



```cpp
// e.
class Master : public GraduateStudent {
private:
    int newId;

public:
    // f.
    Master() : newId(0) {}

    void setNewId(int id) { newId = id; }
    int getNewId() const { return newId; }

    void print() const {
        GraduateStudent::print();
        cout << "New ID: " << newId << endl;
    }
};

int main() {
    // g.
    Student student;
    student.setStudentNumber(19700);
    student.setStudentName("Swekchha Hamal");
    student.setStudentAverage(85.5);
    cout << "Student Information:" << endl;
    student.print();

    Master master;
```

```
~/jpt$ g++ second.cpp -o res2
~/jpt$ ./res2
Student Information:
Student Number: 19700
Student Name: Swekchha Hamal
Student Average: 85.5

Master Information:
Student Number: 111
Student Name: Jasmina Nepal
Student Average: 92.3
Level: 2
Year: 2024
New ID: 9876
~/jpt$
```

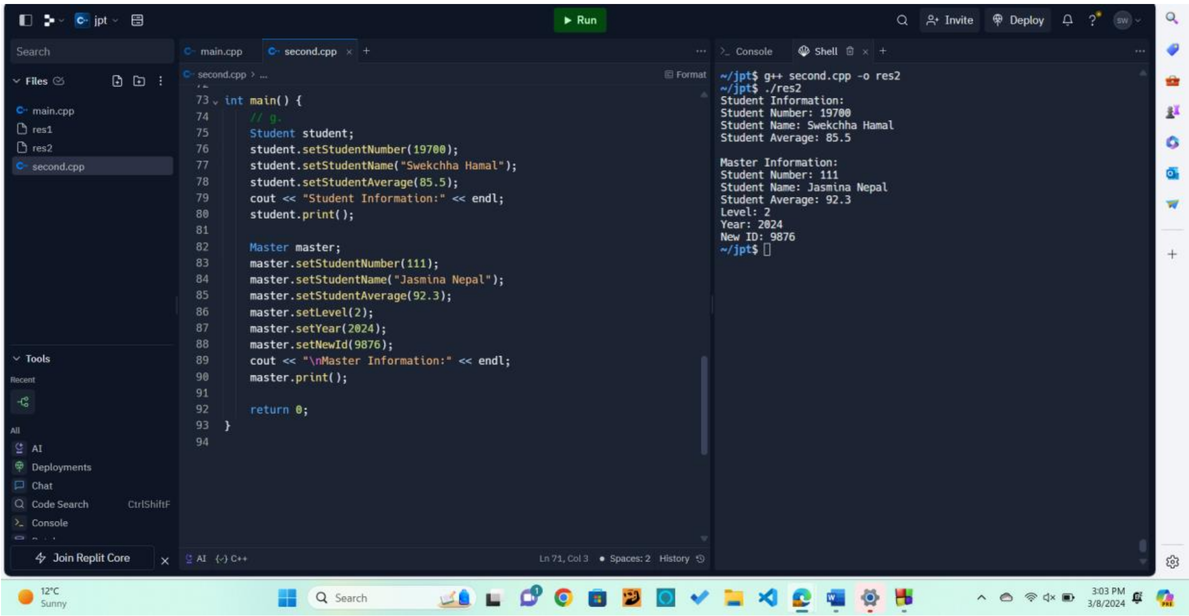
f. Member function:
   i. constructor
   ii. *set* and *get* function for the data member
   iii. *Print* function



```cpp
public:
    // f.
    Master() : newId(0) {}

    void setNewId(int id) { newId = id; }
    int getNewId() const { return newId; }

    void print() const {
        GraduateStudent::print();
        cout << "New ID: " << newId << endl;
    }
};

int main() {
    // g.
    Student student;
    student.setStudentNumber(19700);
    student.setStudentName("Swekchha Hamal");
    student.setStudentAverage(85.5);
    cout << "Student Information:" << endl;
    student.print();

    Master master;
    master.setStudentNumber(111);
    master.setStudentName("Jasmina Nepal");
    master.setStudentAverage(92.3);
    master.setLevel(2);
    master.setYear(2024);
    master.setNewId(9876);
```

```
~/jpt$ g++ second.cpp -o res2
~/jpt$ ./res2
Student Information:
Student Number: 19700
Student Name: Swekchha Hamal
Student Average: 85.5

Master Information:
Student Number: 111
Student Name: Jasmina Nepal
Student Average: 92.3
Level: 2
Year: 2024
New ID: 9876
~/jpt$
```

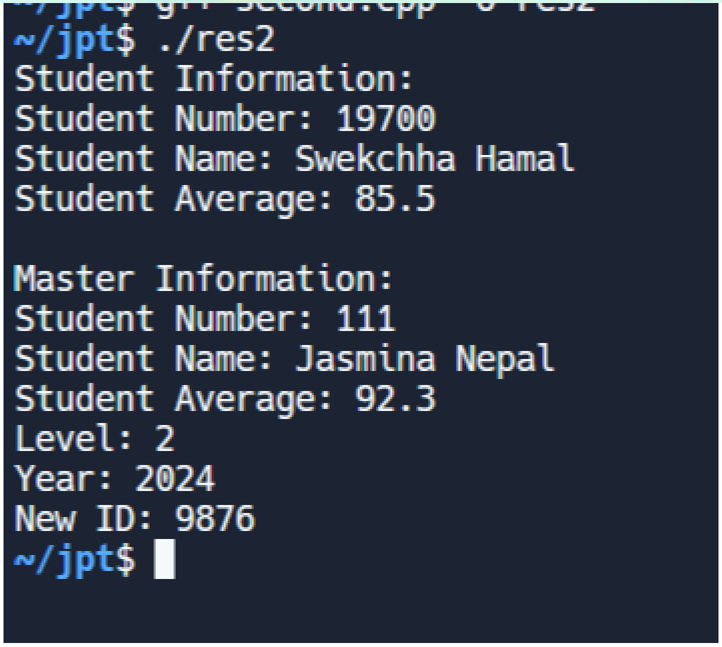
g. Write a driver program(i.e. main function) that:
   i. Declare object of type student with suitable values then print it
   ii. Declare object of type master with your information then print it.



```cpp
int main() {
    // g.
    Student student;
    student.setStudentNumber(19700);
    student.setStudentName("Swekchha Hamal");
    student.setStudentAverage(85.5);
    cout << "Student Information:" << endl;
    student.print();

    Master master;
    master.setStudentNumber(111);
    master.setStudentName("Jasmina Nepal");
    master.setStudentAverage(92.3);
    master.setLevel(2);
    master.setYear(2024);
    master.setNewId(9876);
    cout << "\nMaster Information:" << endl;
    master.print();

    return 0;
}
```

```
~/jpt$ g++ second.cpp -o res2
~/jpt$ ./res2
Student Information:
Student Number: 19700
Student Name: Swekchha Hamal
Student Average: 85.5

Master Information:
Student Number: 111
Student Name: Jasmina Nepal
Student Average: 92.3
Level: 2
Year: 2024
New ID: 9876
~/jpt$
```

Output:

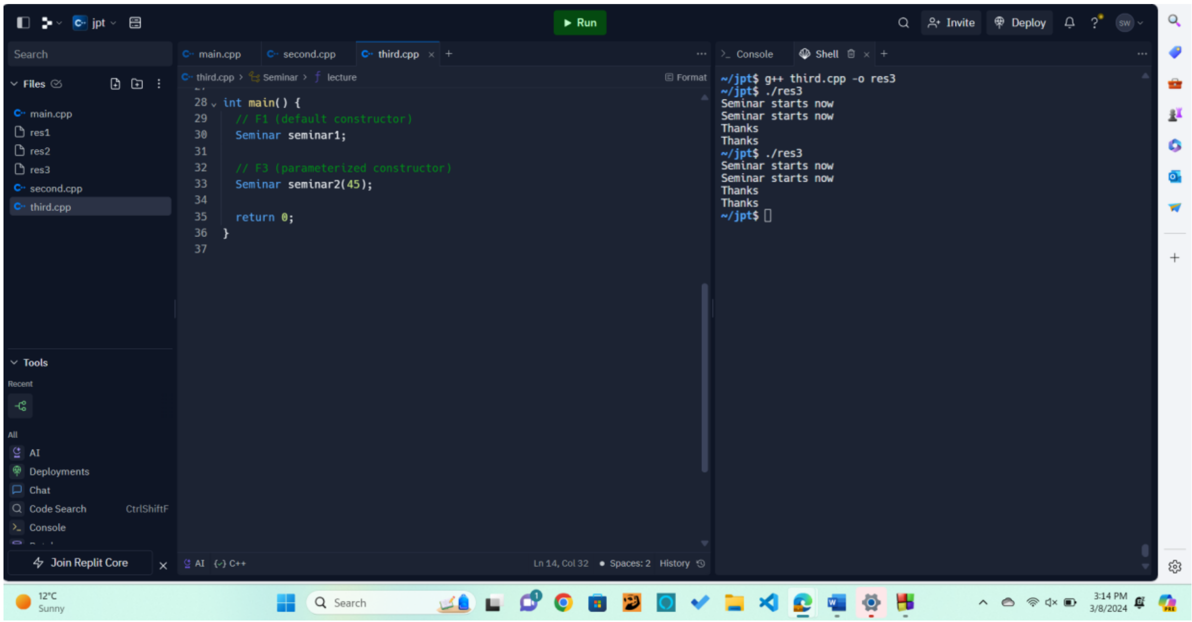


```
~/jpt$ g++ second.cpp -o res2
~/jpt$ ./res2
Student Information:
Student Number: 19700
Student Name: Swekchha Hamal
Student Average: 85.5

Master Information:
Student Number: 111
Student Name: Jasmina Nepal
Student Average: 92.3
Level: 2
Year: 2024
New ID: 9876
~/jpt$
```

3. Answer the questions after going through the following class:

```cpp
class Seminar{
    int time;
    public:
      Seminar()          //Function 1
      {
        time = 30;
        cout << "Seminar starts now" << endl;
      }
      void lecture()          //Function 2
      {
        cout << "Lectures in the seminar on" << endl;
      }
      Seminar(int duration)        //Function 3
      {
        time = duration;
        cout << "Seminar starts now" << endl;
      }
      ~Seminar()        //Function 4
      {
        cout << "Thanks" << endl;
      }
};
```

a. Write statements in C++ that would execute *Function 1* and *Function 3* of class Seminar.



```
int main() {
  // F1 (default constructor)
  Seminar seminar1;

  // F3 (parameterized constructor)
  Seminar seminar2(45);

  return 0;
}
```

```
~/jpt$ g++ third.cpp -o res3
~/jpt$ ./res3
Seminar starts now
Seminar starts now
Thanks
Thanks
~/jpt$ ./res3
Seminar starts now
Seminar starts now
Thanks
Thanks
~/jpt$ 
```

b. In Object Oriented Programming, what is *Function 4* referred as and when does it get invoked/called?

➔ The cleaning function is function 4, which in this instance is the destructor. When a class object is destroyed, either directly by using the delete operator or implicitly when it exits its scope, it is immediately invoked. The destructor's job is to free up any resources the object has allocated before deleting it from memory.

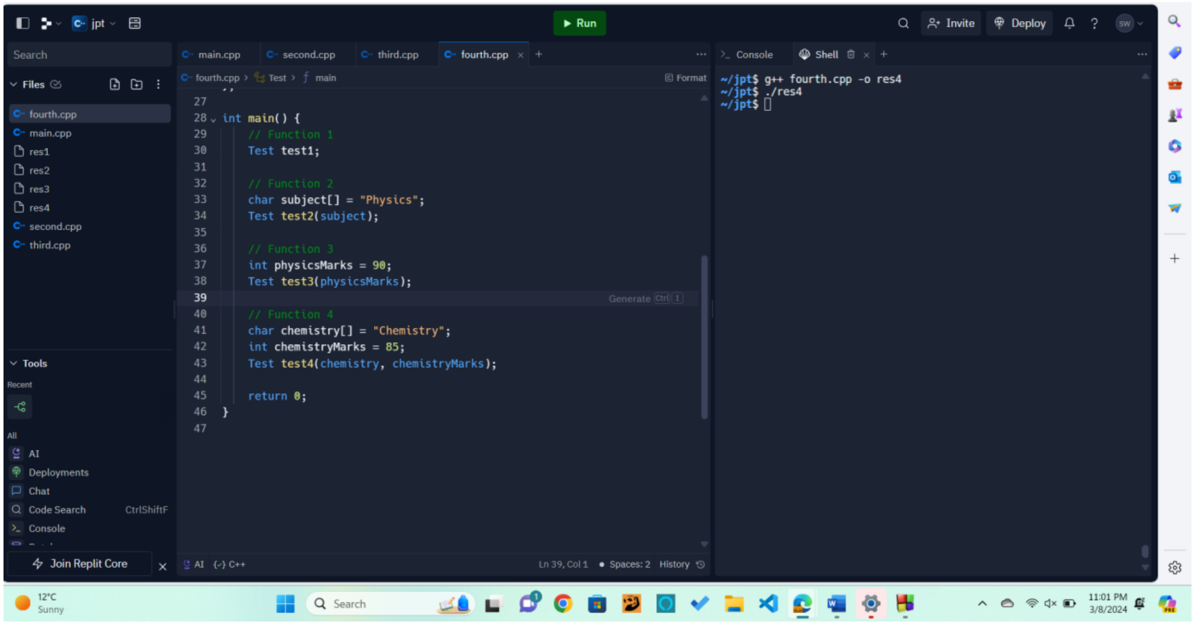
c. In Object Oriented Programming, which concept is illustrated by *Function 1* and *Function 3* together?

➔ Constructor overloading is demonstrated collectively by Functions 1 and 3. Multiple constructors with distinct argument sets can be created within a class thanks to constructor overloading. Here, Function 3 is a parameterized constructor with an integer parameter duration, while Function 1 is the default constructor. To initialize an object of the Seminar class, either Function 1 or Function 3 will be called, depending on how the object is created.

4. Answer the questions after going through the following class:

```
class Test{
    char paper[20];
    int marks;
    public:
      Test ()    // Function 1
      {
        strcpy (paper, "Computer");
        marks = 0;
      }
      Test (char p[])    // Function 2
      {
        strcpy(paper, p);
        marks = 0;
      }
      Test (int m)    // Function 3
      {
        strcpy(paper,"Computer");
        marks = m;
      }
      Test (char p[], int m)    // Function 4
      {
        strcpy (paper, p);
        marks = m;
      }
};
```

a. Write statements in C++ that would execute Fu*nction 1*, *Function 2*, *Function 3* and *Function 4* of class *Test*.



```cpp
};

int main() {
    // Function 1
    Test test1;

    // Function 2
    char subject[] = "Physics";
    Test test2(subject);

    // Function 3
    int physicsMarks = 90;
    Test test3(physicsMarks);

    // Function 4
    char chemistry[] = "Chemistry";
    int chemistryMarks = 85;
    Test test4(chemistry, chemistryMarks);

    return 0;
}
```

```
~/jpt$ g++ fourth.cpp -o res4
~/jpt$ ./res4
~/jpt$ 
```

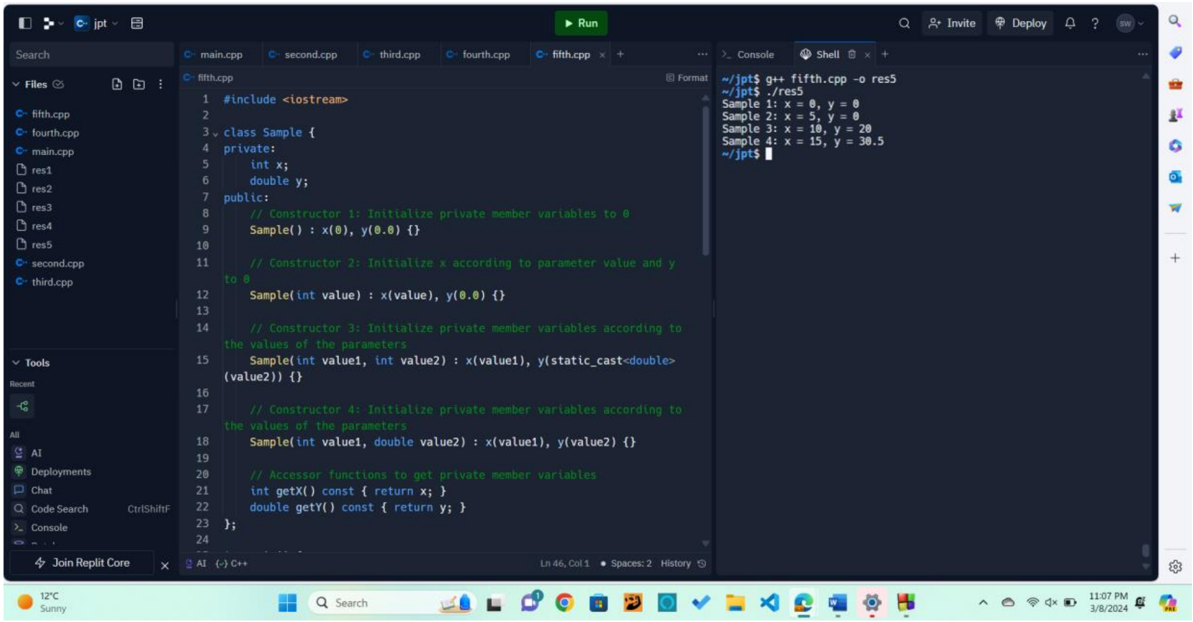
b. Which feature of Object Oriented Programming is demonstrated using *Function 1*, *Function 2*, *Function 3* and *Function 4* together in the above class *Test*?

➔ Constructor overloading is an Object-Oriented Programming characteristic that is shown by Functions 1, 2, 3, and 4 taken collectively in the class test mentioned above. This feature offers versatility in object initialization by enabling the definition of numerous constructors with distinct arguments within a class.

5. Consider the definition of the following class:

```cpp
class Sample{
  private:
    int x;
    double y;
  public :
    Sample(); //Constructor 1
    Sample(int); //Constructor 2
    Sample(int, int); //Constructor 3
    Sample(int, double); //Constructor 4
};
```

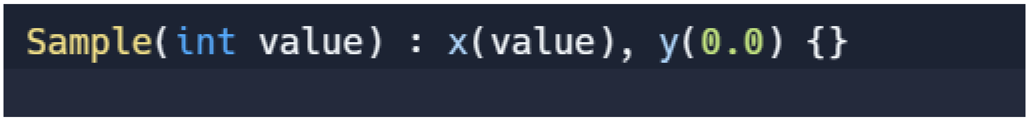
a. Write the definition of the *constructor 1* so that the private member variables are initialized to *0*



```cpp
#include <iostream>

class Sample {
private:
    int x;
    double y;
public:
    // Constructor 1: Initialize private member variables to 0
    Sample() : x(0), y(0.0) {}

    // Constructor 2: Initialize x according to parameter value and y to 0
    Sample(int value) : x(value), y(0.0) {}

    // Constructor 3: Initialize private member variables according to the values of the parameters
    Sample(int value1, int value2) : x(value1), y(static_cast<double>(value2)) {}

    // Constructor 4: Initialize private member variables according to the values of the parameters
    Sample(int value1, double value2) : x(value1), y(value2) {}

    // Accessor functions to get private member variables
    int getX() const { return x; }
    double getY() const { return y; }
};

```

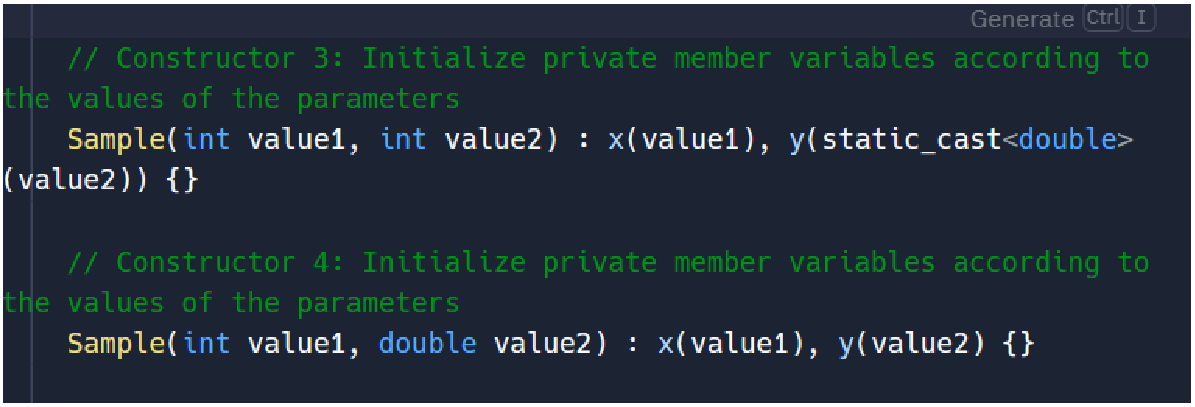
```
~/jpt$ g++ fifth.cpp -o res5
~/jpt$ ./res5
Sample 1: x = 0, y = 0
Sample 2: x = 5, y = 0
Sample 3: x = 10, y = 20
Sample 4: x = 15, y = 30.5
~/jpt$ 
```

b. Write the definition of the *constructor 2* so that the private member variable *x* is initialized according to the value of the parameter, and the private member variable *y* is initialized to *0*



```cpp
Sample(int value) : x(value), y(0.0) {}
```

c. Write the definition of the *constructors 3* and *4* so that the private member variables are initialized according to the values of the parameters.



```cpp
    // Constructor 3: Initialize private member variables according to the values of the parameters
    Sample(int value1, int value2) : x(value1), y(static_cast<double>(value2)) {}

    // Constructor 4: Initialize private member variables according to the values of the parameters
    Sample(int value1, double value2) : x(value1), y(value2) {}
```