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> // Include standard input/output library

#include <iostream> // Include input/output stream library

using namespace std; // Use the standard namespace


class A {

public:

   A(); // Default constructor

   A(int); // Constructor with parameter

   A(const A&); // Copy constructor

   ~A(); // Destructor


public:

   void operator=(const A& rhs); // Overloaded assignment operator

   void Print(); // Non-const print function

   void PrintC() const; // Const print function


   int x; // Public member variable


public:

   int& X() { return x; } // Member function returning reference to x

};


A::A() // Default constructor definition

: x(0) // Initialize x with 0

{

   cout << "Hello from A::A() Default constructor" << endl;

}
```

```
A::A(int i) // Constructor with parameter definition
: x(i) // Initialize x with the given parameter
{
   cout << "Hello from A::A(int) constructor" << endl;
}

A::A(const A& a) // Copy constructor definition
: x(a.x) // Initialize x with the value of x in the passed object
{
   cout << "Hello from A::A(const A&) constructor" << endl;
}

A::~A() // Destructor definition
{
   cout << "Hello from A::A destructor" << endl;
}

void A::operator=(const A& rhs) // Overloaded assignment operator definition
{
   x = rhs.x; // Assign x the value of x in the passed object
   cout << "Hello from A::operator=" << endl;
}

void A::Print() // Non-const print function definition
{
   cout << "A::Print(), x " << x << endl;
}

void A::PrintC() const // Const print function definition
{
   cout << "A::PrintC(), x " << x << endl;
}
```

```
void PassAByValue(A a) // Pass by value function definition
{
   cout << "PassAByValue, a.x " << a.x << endl;
   a.x++; // Increment a's x
   a.Print(); // Call non-const print function
   a.PrintC(); // Call const print function
}

void PassAByReference(A& a) // Pass by reference function definition
{
   cout << "PassAByReference, a.x " << a.x << endl;
   a.x++; // Increment a's x
   a.Print(); // Call non-const print function
   a.PrintC(); // Call const print function
}

void PassAByConstReference(const A& a) // Pass by const reference function definition
{
   cout << "PassAByReference, a.x " << a.x << endl;
   a. PrintC(); // Call const print function
   //a.Print(); // Call to "non-const" print function fails!
   // Compiler error from above line. Why?
}

void PassAByPointer(A* a) // Pass by pointer function definition
{
   cout << "PassAByPointer, a->x " << a->x << endl;
   a->x++; // Increment a's x through pointer
   a->Print(); // Call non-const print function
   a->PrintC(); // Call const print function
}
```

```
int main()
{
    cout << "Creating a0"; getchar();
    A a0; // Call default constructor
    cout << "Creating a1"; getchar();
    A a1(1); // Call constructor with parameter
    cout << "Creating a2"; getchar();
    A a2(a0); // Call copy constructor
    cout << "Creating a3"; getchar();
    A a3 = a0; // Call copy constructor
    cout << "Assigning a3 = a1"; getchar();
    a3 = a1; // Call overloaded assignment operator
    // Call some of the "A" subroutines
    cout << "PassAByValue(a1)"; getchar();
    PassAByValue(a1); // Call pass by value function
    cout << "After PassAByValue(a1)" << endl;
    a1.Print(); // Call non-const print function
    cout << "PassAByReference(a1)"; getchar();
    PassAByReference(a1); // Call pass by reference function
    cout << "After PassAByReference(a1)" << endl;
    a1.Print(); // Call non-const print function
    cout << "PassAByConst(a1)"; getchar();
    PassAByConstReference(a1); // Call pass by const reference function
    cout << "After PassAByConstReference(a1)" << endl;
    a1.Print(); // Call non-const print function
    cout << "PassAByPointer(&a1)"; getchar();
    PassAByPointer(&a1); // Call pass by pointer function
    cout << "After PassAByPointer(a1)" << endl;
    a1.Print(); // Call non-const print function
    cout << "a1.X() = 10"; getchar();
    a1.X() = 10; // Call member function returning reference to x and assign it a value
    a1.Print(); // Call non-const print function
    cout << "PassAByConstReference"; getchar();
    PassAByConstReference(20); // Call pass by const reference function with an integer
    // Why does the above compile? What does it do?
    return 0;
}
```

The program defines a class A with a default constructor, parameterized constructor, copy constructor, destructor, overloaded assignment operator, non-const print function, const print function, and a public member variable x.

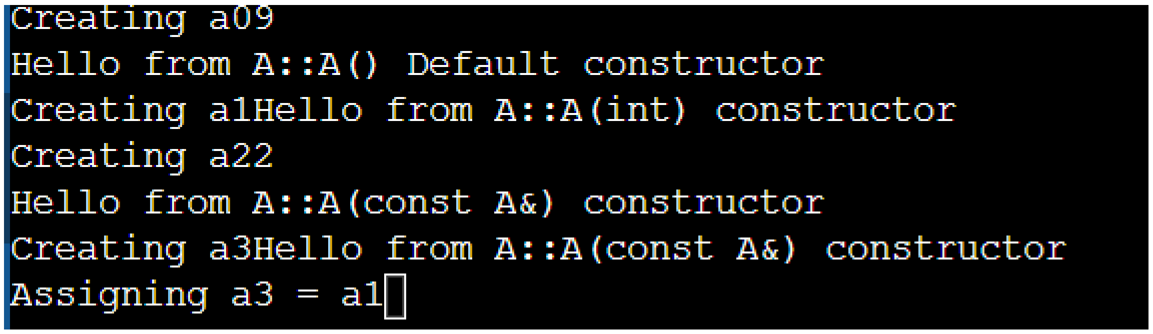
The main function creates instances of class A and calls various member functions and non-member functions to demonstrate pass by value, pass by reference, pass by const reference, and pass by pointer.

The program uses the getchar() function to pause after each step and wait for user input from the standard input device.

Each step is accompanied by a message indicating the action being performed.

The output of each step is printed to the standard output.

The program demonstrates the behavior of different function calls and member function invocations using the class A. Below is output



```
Creating a09
Hello from A::A() Default constructor
Creating a1Hello from A::A(int) constructor
Creating a22
Hello from A::A(const A&) constructor
Creating a3Hello from A::A(const A&) constructor
Assigning a3 = a1
```

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

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.

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

d. Member functions:

i. constructor

ii. set and get functions for each data member

iii. Print function.

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

f. Member function:

i. constructor

ii. set and get function for the data member

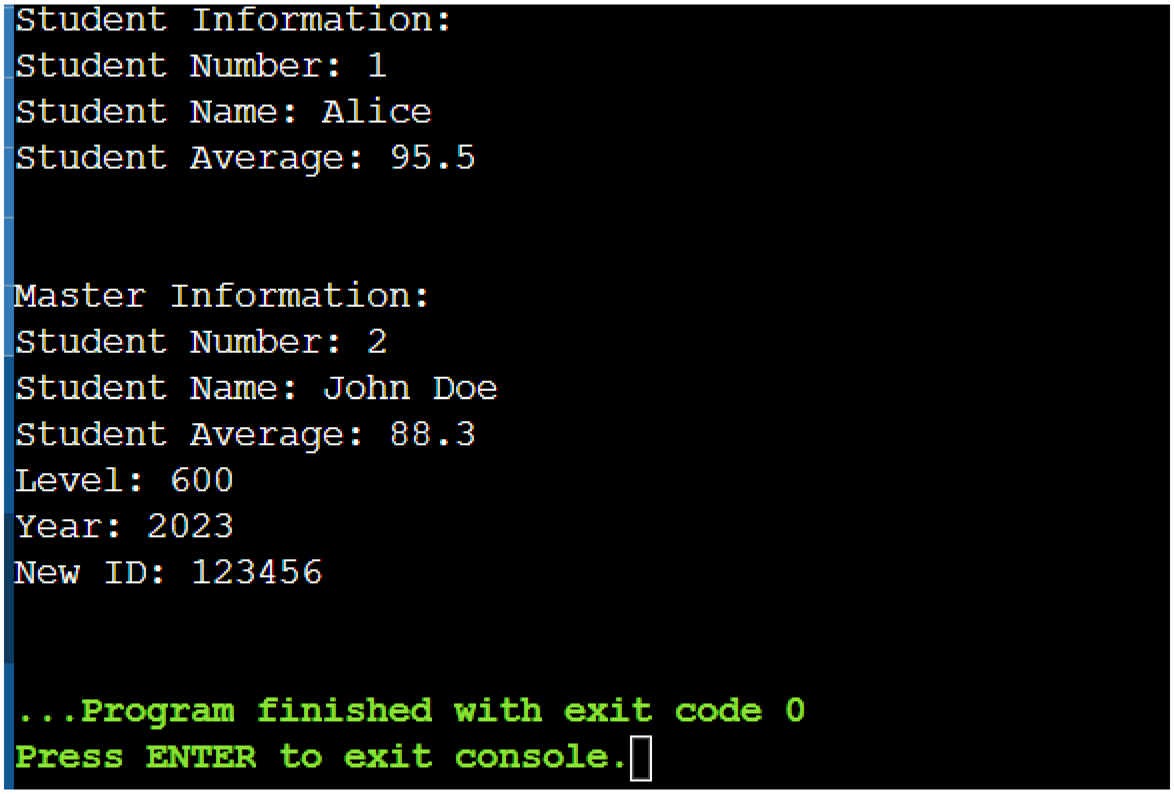
iii. Print function

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

This is the output of the code and the main code is in the cpp file



```
Student Information:
Student Number: 1
Student Name: Alice
Student Average: 95.5


Master Information:
Student Number: 2
Student Name: John Doe
Student Average: 88.3
Level: 600
Year: 2023
New ID: 123456


...Program finished with exit code 0
Press ENTER to exit console.
```

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

```
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.

To execute Function 1 and Function 3 of the Seminar class in C++, you can do the following:

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

int main() {
    Seminar seminar1; // This will execute Function 1

    Seminar seminar2(45); // This will execute Function 3 with duration 45


    return 0;

}
```

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

In Object-Oriented Programming, Function 4 is referred to as the destructor. The destructor is a special member function that gets invoked automatically when an object goes out of scope or is explicitly deleted using the delete keyword. It is used to release resources held by the object, perform cleanup operations, or any other activity that needs to be done before the object's memory is deallocated.

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

Function 1 and Function 3 together illustrate the concept of Constructor overloading. Constructor overloading allows a class to have more than one constructor, each with a different number of parameters or different types of parameters. In this case, Seminar class has two constructors, one without any parameters (Function 1) and one with an integer parameter (Function 3). This allows flexibility in creating objects of the class Seminar with different initializations based on the parameters passed.

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

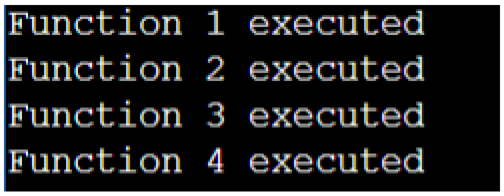
```
class Test{
  char paper[20];
  int marks;
  public:
   Test ()    // Function 1
   {
    strcpy (paper, "Computer");
    marks = 0;
   }
   Test (char p[])    // Function 2
   {
    strcpy(paper, p);
    marks = 0;
   }
   Test (int m)    // Function 3
   {
    strcpy(paper,"Computer");
    marks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

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

a. To execute Function 1, Function 2, Function 3, and Function 4 of the Test class in C++, you can do the following:

```
#include <iostream>

#include <cstring>

using namespace std;


class Test {

private:

    char paper[20];

    int marks;


public:

    Test() {

        strncpy(paper, "Computer", sizeof(paper) - 1);

        paper[sizeof(paper) - 1] = '\0';

        marks = 0;

        cout << "Function 1 executed" << endl;

    }


    Test(char p[]) {

        strncpy(paper, p, sizeof(paper) - 1);

        paper[sizeof(paper) - 1] = '\0';

        marks = 0;

        cout << "Function 2 executed" << endl;

    }


    Test(int m) {

        strncpy(paper, "Computer", sizeof(paper) - 1);
        paper[sizeof(paper) - 1] = '\0';
        marks = m;
        cout << "Function 3 executed" << endl;
    }

    Test(char p[], int m) {
        strncpy(paper, p, sizeof(paper) - 1);
        paper[sizeof(paper) - 1] = '\0';
        marks = m;
        cout << "Function 4 executed" << endl;
    }
};

int main() {
    Test test1;              // Execute Function 1
    Test test2("Math");      // Execute Function 2
    Test test3(80);          // Execute Function 3
    Test test4("Science", 90);  // Execute Function 4

    return 0;
}
```

Output



```
Function 1 executed
Function 2 executed
Function 3 executed
Function 4 executed
```

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*?

The feature of Object-Oriented Programming demonstrated by Function 1, Function 2, Function 3, and Function 4 together in the Test class is Constructor Overloading. Constructor overloading allows a class to have multiple constructors with different types or numbers of parameters. In this case, the Test class has four different constructors, each with a unique set of parameters, allowing objects to be created with various initializations based on the arguments provided. This provides flexibility and customization in object creation within the class.

5. Consider the definition of the following class:

```
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*

```
Sample::Sample() {
  x = 0;
  y = 0.0;
}
```

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*

```
Sample::Sample(int valueX) {
  x = valueX;
  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.

```
Sample::Sample(int valueX, int valueY) {
   x = valueX;
   y = valueY;
}

Sample::Sample(int valueX, double valueY) {
   x = valueX;
   y = valueY;
}
```