Q.No.1)

main.cpp > PassAByReference — Format

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

class A {
public:
    A(); // Default constructor
    A(int); // Parameterized constructor
    A(const A&); // Copy constructor
    ~A(); // Destructor

public:
    void operator=(const A& rhs); // Assignment operator
    void Print(); // Prints the current value of x
    void PrintC() const; // Prints the current value of x, can be called on const objects

    int x; // Public data member

public:
    int& X(); // Returns a reference to x, allowing for direct modification
};

// Definitions of A's members
A::A() : x(0) {
    cout << "Hello from A::A() Default constructor" << endl;
}
```

```
~/assg3cs360$ g++ main.cpp -o test2
~/assg3cs360$ ./test2
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
PassAByConstReference(a1)
PassAByConstReference, 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
```

- class A: Defines a class named A with one public data member x, a constructor, a copy constructor, an assignment operator, a destructor, and member functions Print, PrintC, and X.
- A(), A(int), A(const A&): These are three constructors for class A. The first is the default constructor, initializing x to 0. The second takes an integer to initialize x. The third is the copy constructor, which initializes a new object with the same x value as an existing object.
- ~A(): The destructor, which is called when an instance of A is destroyed.
- void operator=(const A& rhs): The assignment operator, used to copy the value from one object to another of the same class.
- void Print() and void PrintC() const: Member functions to print the value of x. PrintC is a constant member function, meaning it guarantees not to modify the object.
- int& X(): Returns a reference to x, allowing direct modification of x.

The line PassAByConstReference(20); in the code snippet compiles because the A class must have a constructor that can take an int as an argument. This constructor is not marked explicit, which allows the compiler to implicitly convert the int to an object of type A when the function PassAByConstReference is called.

Q.No.2)

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

class student {
protected:
    int studentNumber;
    string studentName;
    double studentAverage;

public:
    student() : studentNumber(0), studentName(""), studentAverage(0.0) {}

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

    void setStudentName(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;
```

Run

```
Student Information:
Student Number: 1
Student Name: Ronish Shrestha
Student Average: 89.5

Master Student Information:
Student Number: 2
Student Name: John Wick
Student Average: 91.2
Level: 2
Year: 2022
New ID: 12345
```

Q.no.3)

a)

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

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;
    }
};

int main() {
```

```
Seminar starts now
Seminar starts now
Thanks
Thanks
```

b)
In Object Oriented Programming, Function 4, as found in the class Seminar provided earlier, is referred to as a destructor. A destructor is a special member function that is invoked when an object goes out of scope or is explicitly deleted. It has the same name as the class but is prefixed with a tilde (~). It is used to clean up resources that the object may have acquired during its lifetime. The destructor is called automatically by the C++ runtime system.

c)

The concept illustrated by Function 1 (the default constructor) and Function 3 (the parameterized constructor) together is "Constructor Overloading." In Object-Oriented Programming, constructor overloading refers to having multiple constructors within a class, each with a different number or type of parameters. This allows objects of the class to be initialized in different ways.

- The default constructor (Function 1) is a constructor with no parameters, used to initialize objects with default values.
- The parameterized constructor (Function 3) takes one or more parameters, allowing for the initialization of objects with specific values.

4.

a)

```cpp
#include <iostream>
#include <cstring> // For strcpy

using namespace std;

class Test {
    char paper[20];
    int marks;

public:
    // Function 1: Default constructor
    Test() {
        strcpy(paper, "Computer");
        marks = 0;
    }

    // Function 2: Constructor with a char array parameter
    Test(char p[]) {
        strcpy(paper, p);
        marks = 0;
    }

    // Function 3: Constructor with an int parameter
    Test(int m) {
        strcpy(paper, "Computer");
        marks = m;
    }
```

```
Paper: Computer, Marks: 0
Paper: Mathematics, Marks: 0
Paper: Computer, Marks: 89
Paper: Physics, Marks: 88
```

b)

The feature of Object-Oriented Programming demonstrated using Function 1, Function 2, Function 3, and Function 4 together in the class Test is Constructor Overloading. Constructor overloading allows a class to have multiple constructors with the same name but different parameters. This enables the creation of objects with different initial states, depending on the arguments passed to the constructors. It is a type of polymorphism where multiple constructors perform different tasks using the same name but with different lists of parameters.

5a)

main.cpp > ... Format

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

class Sample {
private:
    int x;
    double y;
public:
    // Constructor 1
    Sample() : x(0), y(0.0) {
        // Member initializer list sets both x and y to 0
    }

    // Constructor 2
    Sample(int xValue) : x(xValue), y(0.0) {
        // Member initializer list sets x to the argument xValue and y to 0
    }

    // Constructor 3
```

Run

```
s1: x: 0, y: 0
s2: x: 10, y: 0
s3: x: 20, y: 30
s4: x: 40, y: 50.5
```

```cpp
public:
    // Constructor 1
    Sample() : x(0), y(0.0) {
        // Member initializer list sets both x and y to 0
    }

    // Constructor 2
    Sample(int xValue) : x(xValue), y(0.0) {
        // Member initializer list sets x to the argument xValue and y to 0
    }
```

5c)

```cpp
    // Constructor 3
    Sample(int xValue, int yValue), y(yValue) {
        // Member initializer list sets x to xValue and y to yValue (int converted to double)
    }

    // Constructor 4
    Sample(int xValue, double yValue) : x(xValue), y(yValue) {
        // Member initializer list sets x to xValue and y to yValue
    }
```