LAB NO. 07

CONSTRUCTORS MEMBER INITIALIZATION LIST IN C++

Lab Objectives

Following are the lab objectives:

1. Initializer list in C++

Instructions

- This is individual Lab work/task.
- Complete this lab work within lab timing.
- Discussion with peers is not allowed.
- Copy paste from Internet will give you **negative marks**.
- Lab work is divided into small tasks, complete all tasks sequentially.

Initializer List in C++

In the previous lab, we learned about how classes and their objects can be created and the different ways their members can be accessed. We also saw how data members are initialized in the constructor of a class as shown below.

```
class Rectangle
{
   int length;
   int breadth;
   public:
      Rectangle()
      {
         length = 7;
         breadth = 4;
      }
};
```

Syntax:

```
Constructorname(datatype value1, datatype value2):datamember(value1),datamember(value2)
{
   ...
}
```

In the above constructor, we assigned the values 7 and 4 to the data members length and breadth respectively. Note that, here we assigned the values to these variables, not initialized.

In the case of constructors having parameters, we can directly initialize our data members using initialization lists.

Using initialization list, we can write the above code as follows.

```
class Rectangle
{
   int length;
   int breadth;
   public:
      Rectangle() : length(7), breadth(4) // initializing data members
      {
         // no need to assign anything here
      }
```

```
};
```
An initializer list starts after the constructor name and its parameters and begins with a colon (:) followed by the list of variables which are to be initialized separated by a comma with their values in curly brackets.

Let's see an example for the above code.

```
#include <iostream>
using namespace std;
class Rectangle
{
	int length;
	int breadth;
	public:
		Rectangle() : length(7), breadth(4)
		{

		}
		int printArea()
		{
			return length * breadth;
		}
};
int main()
{
	Rectangle rt;
	cout << rt.printArea() << endl;
	return 0;
}
```

Let's see another example of initialization list in case of parameterized constructor.

```
#include <iostream>
using namespace std;
class Rectangle
{
	int length;
	int breadth;
	public:
		Rectangle( int l, int b ) : length(l), breadth(b)
```

```
        {

        }
        int printArea()
        {
                return length * breadth;
        }
};
int main()
{
        Rectangle rt( 7, 4 );
        cout << rt.printArea() << endl;
        return 0;
}
```

In this example, the initializer list is directly initializing the variables length and breadth with the values of l and b which are 7 and 4 respectively.

This is the same as the following code with the only difference that in the above example, we are directly initializing the variables while in the following code, the variables are being assigned the parameterized values.

```
class Rectangle
{
   int length;
   int breadth;
   public:
      Rectangle( int l, int b )
      {
         length = l;
         breadth = b;
      }
};
```

Need for Initialization List

1) For initializing const data member

Though we can use initialization list anytime as we did in the above examples, but there are certain cases where we have to use initialization list otherwise the code won't work.

If we declare any variable as const, then that variable can be initialized, but not assigned.

Any variable declared with const keyword before its data type is a const variable.

To initialize a const variable, we use initialization list. Since we are not allowed to assign any value to a const variable, so we cannot assign any value in the body of the constructor. So, this is the case where we need initialization list.

Following is an example initializing a const data member with initialization list.

```
#include <iostream>
using namespace std;
class Rectangle
{
        const int length;
        const int breadth;
        public:
                Rectangle( int l, int b ) : length(l), breadth(b)
                {

                }
                int printArea()
                {
                        return length * breadth;
                }
};
int main()
{
        Rectangle rt( 7, 4 );
        cout << rt.printArea() << endl;
        return 0;
}
```

2) When data member and parameter have same name

```
#include<iostream>
using namespace std;
class Base
{
   private:
   int value;
   public:
   Base(int value):value(value)
   {
      cout << "Value is " << value;
```

```
    }
};
int main()
{
    Base il(10);
    return 0;
}
```

Lab Tasks

Q1). Suppose you have an account in HBL Bank with an initial amount of 10k and you have to add some more amount to it. Create a class 'AddAmount' with a data member named 'amount' with an initial value of 10k. Now make two constructors of this class as follows:

1. Create a default constructor that uses a member initializer list that allows the user to initialize initial value in account.
2. having a parameter which is the amount that will be added to the account.

Create an object of the 'AddAmount' class and display the final amount in the account.

Q2). Declare a class **Area** which perform the following tasks using constructor chaining:

1. Create a default constructor that uses a member initializer list that initialize the data members (length, width and breadth)
2. Calculate area of rectangle when length and width are passed.
3. Calculate area of circle when radius is passed.
4. Calculate area of cube when length, width and breadth is passed.
5. Define destructor at the end of Area class.