**Object Oriented Programming Lab 04**

|  |  |
| --- | --- |
| **Course**: Object Oriented Programming (CL1004) | **Semester**: Spring 2025 |
| **Instructor**: Shafique Rehman |  |
| Note:   * Maintain discipline during the lab. * Listen and follow the instructions as they are given. * Just raise hand if you have any problem. * Completing all tasks of each lab is compulsory. * Get your lab checked at the end of the session. |  |

**Constructor**

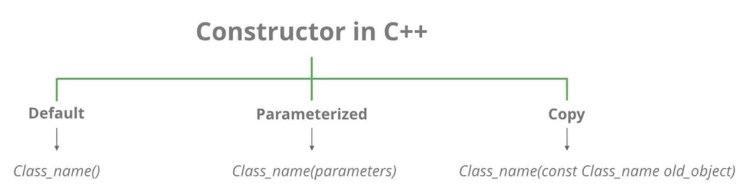
Constructor is the special type of member function in C++ classes. It is automatically invoked when an object is being created. It is special because its name is same as the class name. A constructor can be used for the following functions:

* **To initialize data member of class:** In the constructor member function (which the programmer will declare), we can initialize the default vales to the data members and they can be used further for processing.
* **To allocate memory for data member:** Constructor is also used to declare run time memory (dynamic memory for the data members).

Constructor has the following properties:

* Constructor has the same name as the class name.
* The name is case sensitive.
* Constructor does not have return type.
* We can overload constructor; it means we can create more than one constructor of class (we will discuss this in later labs).
* It must be public type (declared inside the public access modifier in the class).

**Types of Constructor**

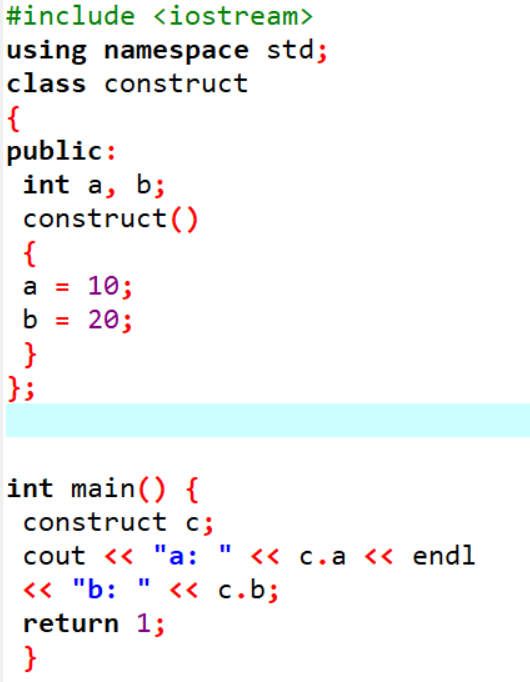


Let us understand the types of constructors in C++ by taking a real-world example. Suppose you went to a shop to buy a marker. When you want to buy a marker, what are the options. The first one you go to a shop and say give me a marker. So just saying give me a marker mean that you did not set which brand name and which color, you didn’t mention anything just say you want a marker. So, when we said just I want a marker so whatever the frequently sold marker is there in the market or in his shop he will simply hand over that. And this is what a default constructor is!

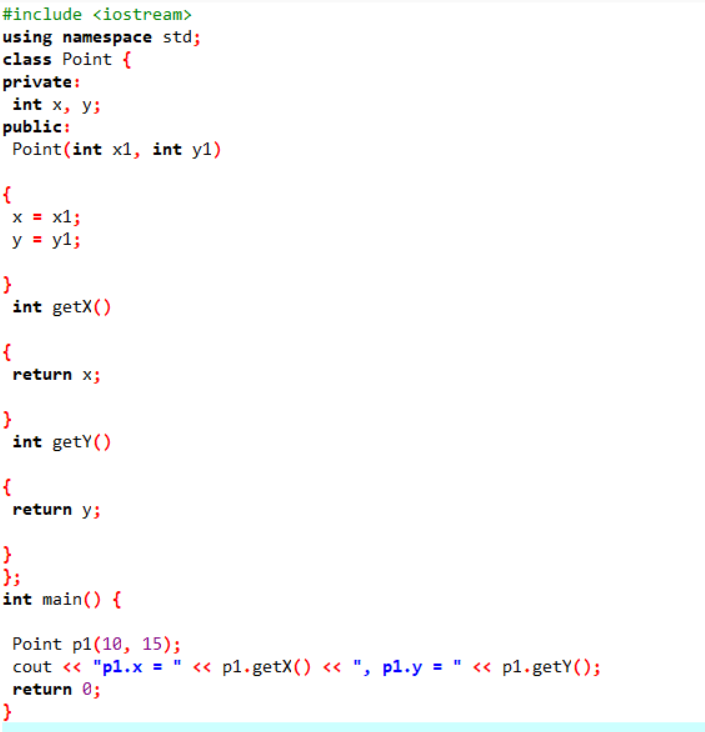
The second method is you go to a shop and say I want a marker a red in color and XYZ brand. So, you are mentioning this and he will give you that marker. So, in this case you have given the parameters and this is what a parameterized constructor is! Then the third one you go to a shop and say I want a marker like this (a physical marker on your hand). So, the shopkeeper will see that marker. Okay, and he will give a new marker for you. So copy of that marker. And that’s what a copy constructor is!

* **Default Constructors:** Default constructor is the constructor, which does not take any argument. It has no parameters.
* **Null constructors:** Null constructors in C++ are a special type of constructor that does nothing. The compiler knows that there is no code to execute, so it will not generate any executable code for the constructor.
* **Parameterized Constructors:** It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor’s body, use the parameters to initialize the object.
* **Copy Constructor:** A copy constructor is a member function, which initializes an object using another object of the same class. The copy constructor in C++ is used to copy data of one object to another.

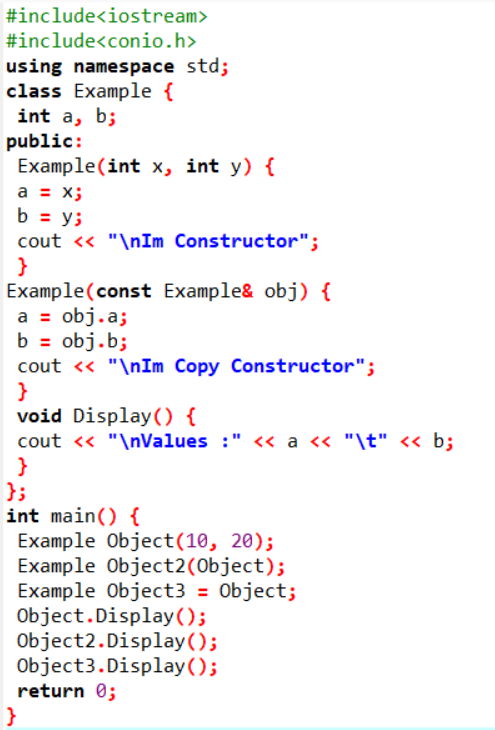
**Example: Default Constructor**



**Example: Parameterized Constructor**



**Example: Copy Constructor**

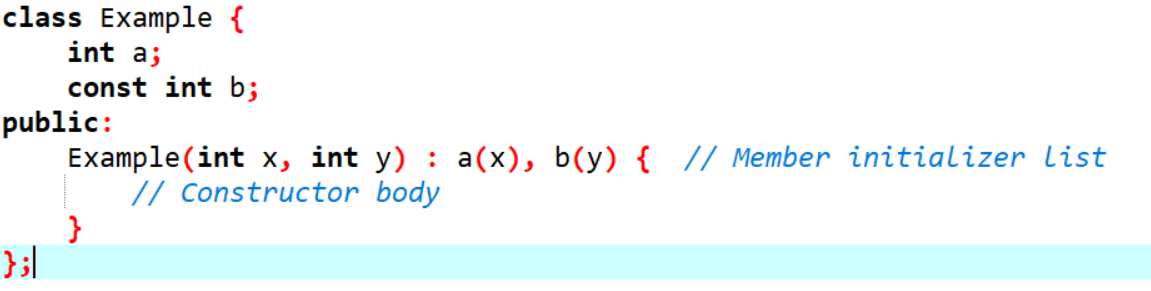


**Member Initializer List**

In C++, a member initializer list is used to initialize class members directly when an object is created. It is written after the constructor’s parameter list and before the constructor’s body. This approach is especially useful for constant variables, reference variables, and base class initialization. Instead of assigning values inside the constructor body, we can directly initialize the variables in the initializer list. It is written after the constructor’s parameter list and before the constructor body.

This method is very useful for initializing constant variables, reference variables, and base class members because they must be set at the time of object creation. It also helps in making the code faster and more efficient by avoiding extra assignments inside the constructor.

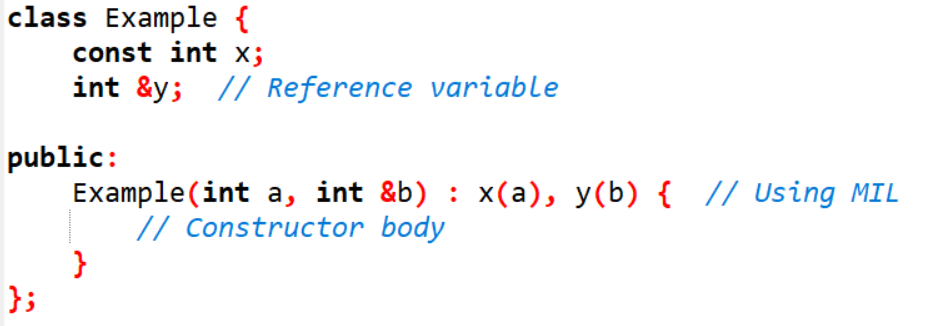
Consider the example below:



Here, **a** and **b** are initialized using the member initializer list before entering the constructor body.

**Advantages**

* Improves performance by avoiding extra assignments.
* Required for const and reference members since they must be initialized at declaration.

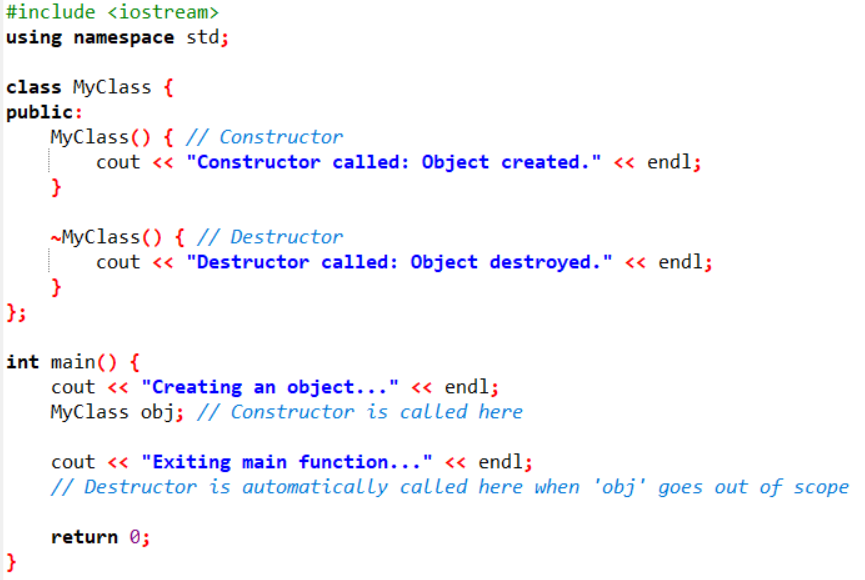


* Ensures base class constructors are called properly in inheritance (will be discussed later in future labs).

**Destructor**

A destructor is also a special member function as a constructor. Destructor destroys the class objects created by the constructor. Destructor has the same name as their class name preceded by a tilde (~) symbol. It is not possible to define more than one destructor. The destructor is only one way to destroy the object created by the constructor. Hence destructor can-not be overloaded. Destructor neither requires any argument nor returns any value. It is automatically called when the object goes out of scope. Destructors release memory space occupied by the objects created by the constructor. In destructor, objects are destroyed in the reverse of object creation.

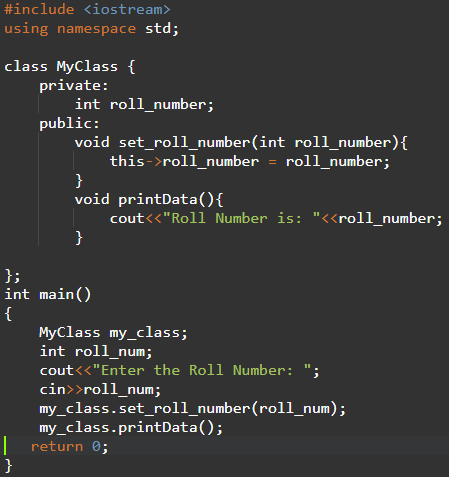
**Example: Destructor**



**This Keyword**

This keyword is a pointer to the current object. It is used to refer to the member variables and member functions of the current object. It can be particularly useful when there is a naming conflict between a member variable and a local variable or parameter with the same name in a member function. By using this, you can disambiguate between the two and access the member variable specifically.

Consider the below example:



In this example, the MyClass class has a private member variable roll\_number. The set\_ roll\_number member function takes an integer parameter roll\_number and sets the value of the member variable to it using the this keyword. The printData member function simply prints the value of the member variable to the console, again using the this keyword to refer to the member variable.

**Let`s Explore:**

1. What will happen if we will not use “this pointer” in above example?
2. Which value will be assigned to member variable roll\_number? And why?
3. Which value will be assigned to local variable roll\_number? And why?

**Rule of Three**

If you need to explicitly declare either the destructor, copy constructor or copy assignment operator yourself, you probably need to explicitly declare all three of them. The default constructors and assignment operators do shallow copy and we create our own constructor and assignment operators when we need to perform a deep copy (For example when a class contains pointers pointing to dynamically allocated resources).

First, what does a destructor do? It contains code that runs whenever an object is destroyed. Only affecting the contents of the object would be useless. An object in the process of being destroyed cannot have any changes made to it. Therefore, the destructor affects the program’s state as a whole. Now, suppose our class does not have a copy constructor. Copying an object will copy all of its data members to the target object. In this case when the object is destroyed the destructor runs twice. Also, the destructor has the same information for each object being destroyed. In the absence of an appropriately defined copy constructor, the destructor is executed twice when it should only execute once. This duplicate execution is a source for trouble.

