Order of Constructor Call with Inheritance

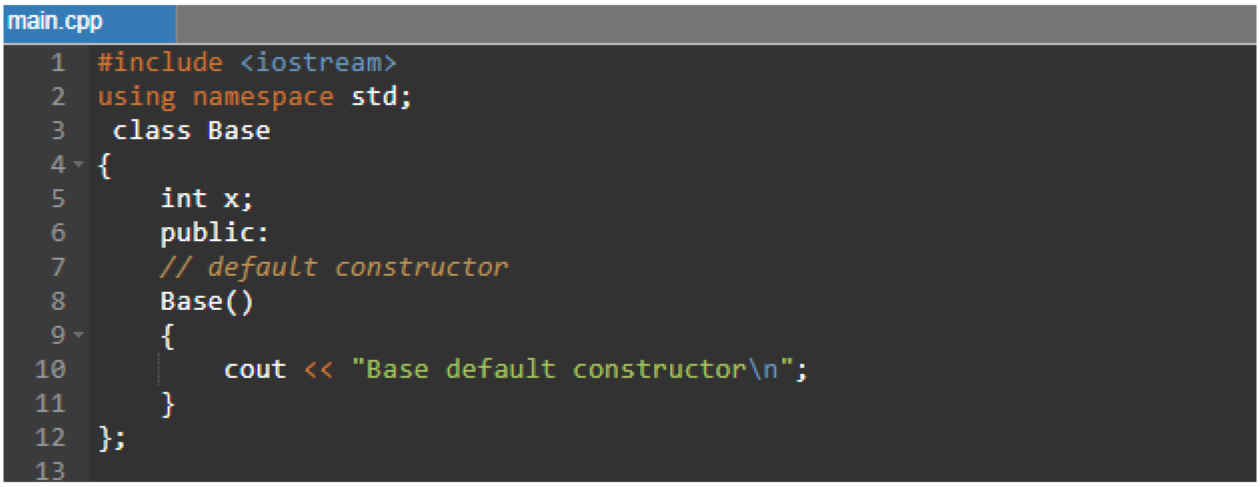
Base class constructors are always called in the derived class constructors. Whenever you create derived class object, first the base class default constructor is executed and then the derived class's constructor finishes execution.

Points to Remember

1. Whether derived class's default constructor is called or parameterised is called, base class's default constructor is always called inside them.
2. To call base class's parameterised constructor inside derived class's parameterised constructor, we must mention it explicitly while declaring derived class's parameterized constructor.

Base class Default Constructor in Derived class Constructors

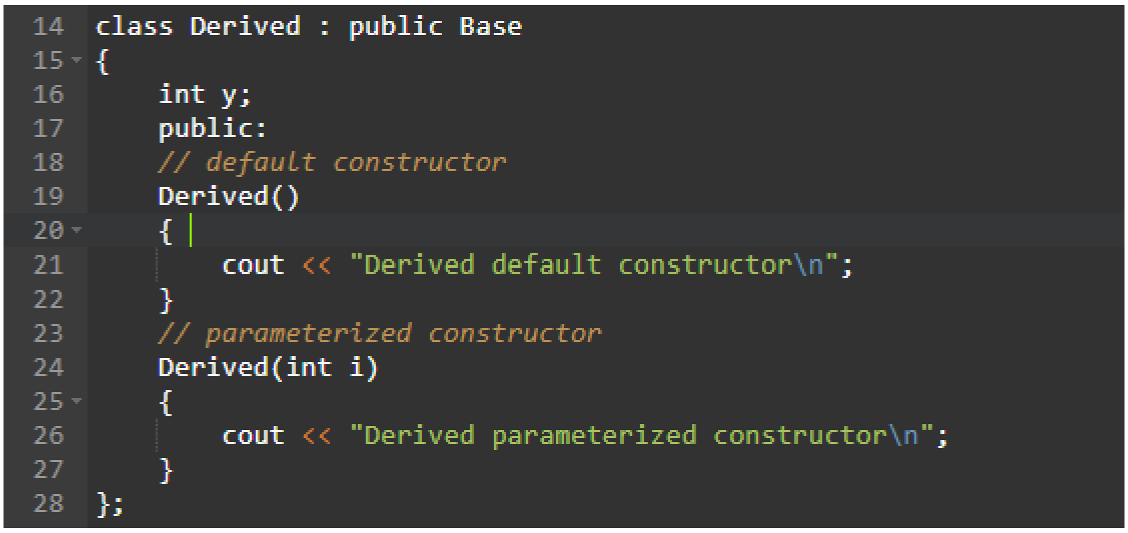
Default constructor is present in all the classes. In the below example we will see when and why Base class's and Derived class's constructors are called.



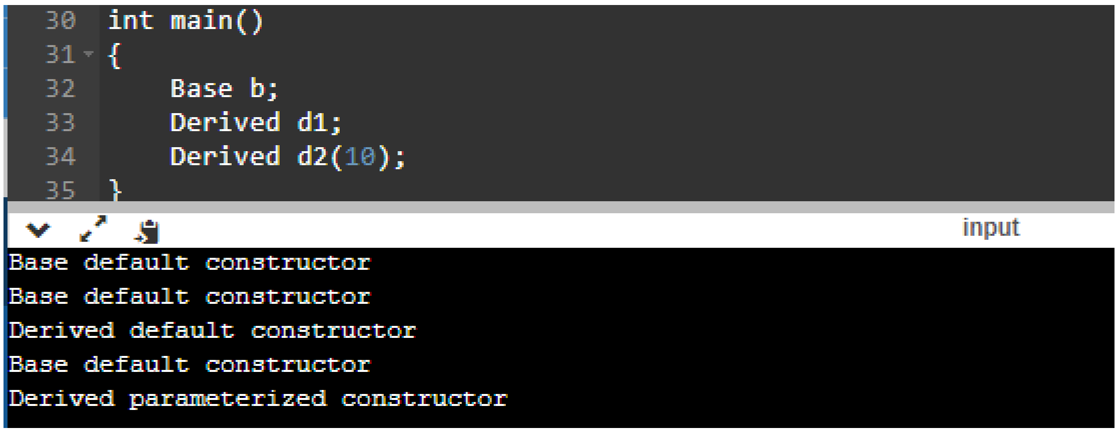
main.cpp

```cpp
#include <iostream>
using namespace std;
 class Base
{
    int x;
    public:
    // default constructor
    Base()
    {
        cout << "Base default constructor\n";
    }
};

```



```cpp
class Derived : public Base
{
    int y;
    public:
    // default constructor
    Derived()
    {
        cout << "Derived default constructor\n";
    }
    // parameterized constructor
    Derived(int i)
    {
        cout << "Derived parameterized constructor\n";
    }
};
```

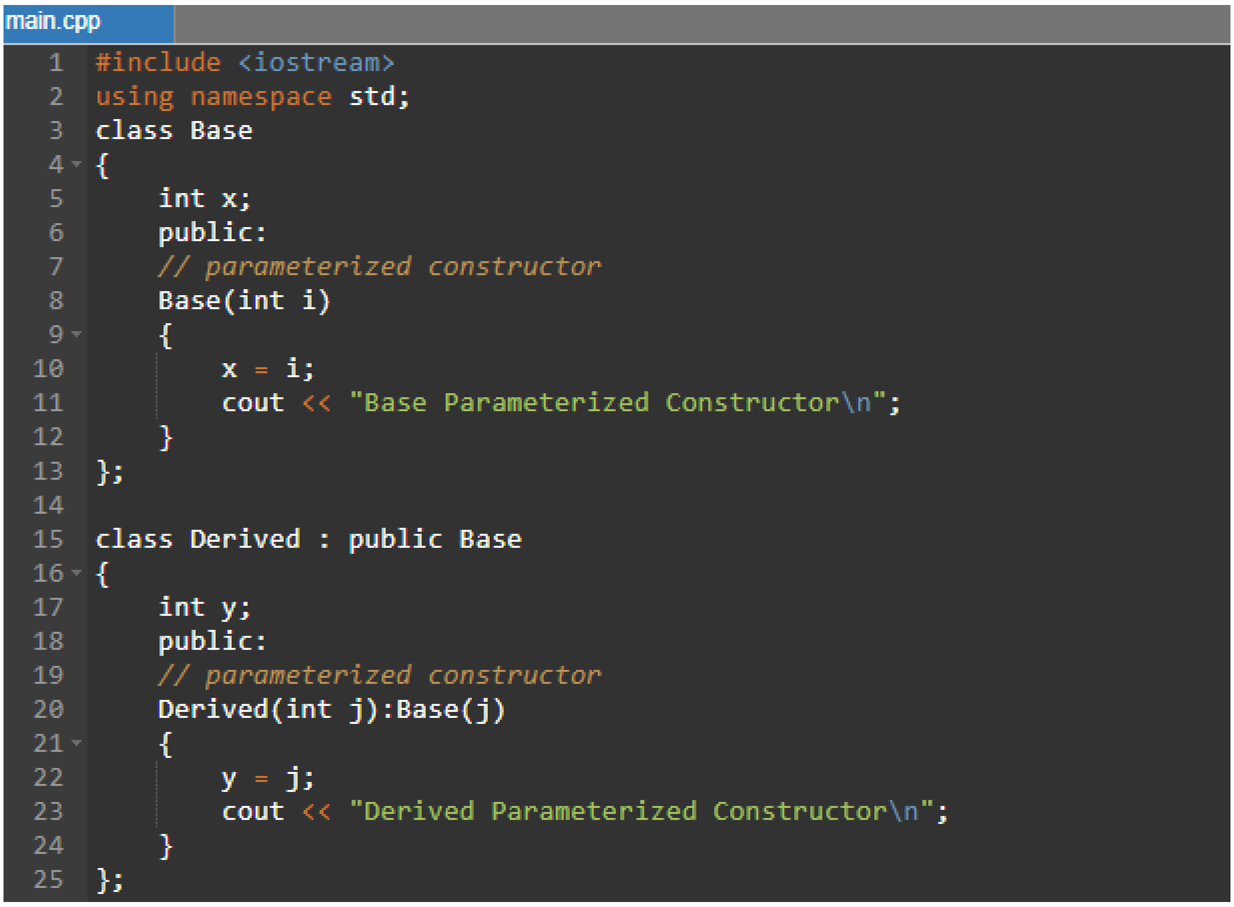


```
int main()
{
    Base b;
    Derived d1;
    Derived d2(10);
}
```

```
Base default constructor
Base default constructor
Derived default constructor
Base default constructor
Derived parameterized constructor
```

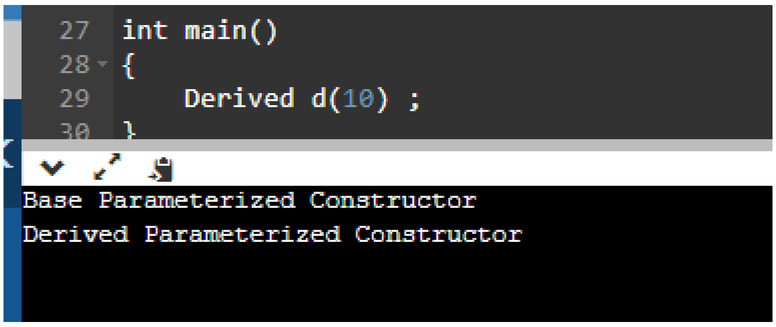
You will see in the above example that with both the object creation of the Derived class, Base class's default constructor is called.

Base class Parameterized Constructor in Derived class Constructor



```
#include <iostream>
using namespace std;
class Base
{
    int x;
    public:
    // parameterized constructor
    Base(int i)
    {
        x = i;
        cout << "Base Parameterized Constructor\n";
    }
};

class Derived : public Base
{
    int y;
    public:
    // parameterized constructor
    Derived(int j):Base(j)
    {
        y = j;
        cout << "Derived Parameterized Constructor\n";
    }
};
```



```
int main()
{
    Derived d(10) ;
}
```

```
Base Parameterized Constructor
Derived Parameterized Constructor
```

Why is Base class Constructor called inside Derived class?

Constructors have a special job of initializing the object properly. A Derived class constructor has access only to its own class members, but a Derived class object also have inherited property of Base class, and only base class constructor can properly initialize base class members. Hence all the constructors are called, else object wouldn't be constructed properly.

Constructor call in Multiple Inheritance in C++

Its almost the same, all the Base class's constructors are called inside derived class's constructor, in the same order in which they are inherited.

```
class A : public B, public C ;
```

In this case, first class B constructor will be executed, then class C constructor and then class A constructor.

Functions that are never Inherited

- Constructors and Destructors are never inherited and hence never overrided.(We will study the concept of function overriding in the next tutorial)
- Also, assignment operator `=` is never inherited. It can be overloaded but can't be inherited by sub class.

Hybrid Inheritance and Constructor call

As we all know that whenever a derived class object is instantiated, the base class constructor is always called. But in case of Hybrid Inheritance, as discussed in above example, if we create an instance of class D, then following constructors will be called :

- before class D's constructor, constructors of its super classes will be called, hence constructors of class B, class C and class A will be called.

- when constructors of class B and class C are called, they will again make a call to their super class's constructor.

This will result in multiple calls to the constructor of class A, which is undesirable. As there is a single instance of virtual base class which is shared by multiple classes that inherit from it, hence the constructor of the base class is only called once by the constructor of concrete class, which in our case is class D.

If there is any call for initializing the constructor of class A in class B or class C, while creating object of class D, all such calls will be skipped.