Subject Name: Object Oriented Programming Using C++

Subject Code: BCA-301 N

Subject Topic: Order of Constructor Call in Inheritance

Abhishek Dwivedi

Assistant Professor

Department of Computer Application

UIET, CSJM University, Kanpur

Order of Constructor Call with Inheritance in C++

- Whenever we create an object of a class, the default constructor of that class is invoked automatically to initialize the members of the class. If we inherit a class from another class and create an object of the derived class, it is clear that the default constructor of the derived class will be invoked but before that the default constructor of all of the base classes will be invoke, i.e the order of invocation is that the base class's default constructor will be invoked first and then the derived class's default constructor will be invoked.

Reason behind the base class’s constructor is called on creating an object of derived class

- What happens when a class is inherited from other? The data members and member functions of base class comes automatically in derived class based on the access specifiers but the definition of these members exists in base class only. So when we create an object of derived class, all of the members of derived class must be initialized but the inherited members in derived class can only be initialized by the base class’s constructor as the definition of these members exists in base class only. This is why the constructor of **base class is called first to initialize all the inherited members.**

Example

```
class Parent
{
    public:
    Parent()
    {
        cout << "Inside base class" << endl;
    }
};

class Child : public Parent
{
    public:
    Child()
    {
        cout << "Inside sub class" << endl;
    }
};
```

```
void main()
{
    Child obj;          // creating object of sub class
    getch();
}
```

- Output:

```
Inside base class
Inside sub class
```

Order of constructor call for Multiple Inheritance

- For multiple inheritance order of constructor call is, the base class’s constructors are called in the order of inheritance and then the derived class’s constructor.

```
class Parent1
{
     public:
    Parent1()
    {
        cout << "Inside first base class" << endl;
    }
};

class Parent2
{
    public:
    Parent2()
    {
        cout << "Inside second base class" << endl;
    }
};
```

```
class Child : public Parent1, public Parent2
{
    public:
    Child()
    {
        cout << "Inside child class" << endl;
    }
};
void main()
{
    Child obj1;
    getch();
}
```

Order of constructor and Destructor call for a given order of Inheritance

Order of Inheritance

$$\begin{tikzcd}
\text{Class C} & \text{(Base Class 2)} \\
\text{Class B} \arrow[from=1-1, d] & \text{(Base Class 1)} \\
\text{Class A} \arrow[from=2-1, d] & \text{(Derived Class)}
\end{tikzcd}$$

Order of Constructor Call

1. C() (Class C's Constructor)
2. B() (Class B's Constructor)
3. A() (Class A's Constructor)

Order of Destructor Call

1. ~A() (Class A's Destructor)
2. ~B() (Class B's Destructor)
3. ~C() (Class C's Destructor)

Calling the parameterized constructor of base class in derived class constructor

- To call the parameterized constructor of base class when derived class's parameterized constructor is called, you have to explicitly specify the base class's parameterized.
- Syntax for derived class constructor:

```
Derived_constructor (arglist1,arglist2,……arglistN, arglistof_derivedclass) :
    base1(arglist1),
    base2(arglist2),
    …….
    baseN(arglistN),
{
    //Body of derived constructor
}
```

```
class Parent
{
   public:
   Parent(int i)
   {
     int x =i;
       cout << "Inside base class's parameterized
   constructor" << endl;
   }
};
```

```
class Child : public Parent
{
   public:

  Child(int j): Parent(j) // explicitly mention to call the Base class's parameterized constructor

        {
        cout << "Inside sub class's parameterized constructor" << endl;
        }
};

void main()
 {
   Child obj1(10);
   getch();
}
```

- **Output:**

1. Inside base class's parameterized constructor
2. Inside sub class's parameterized constructor

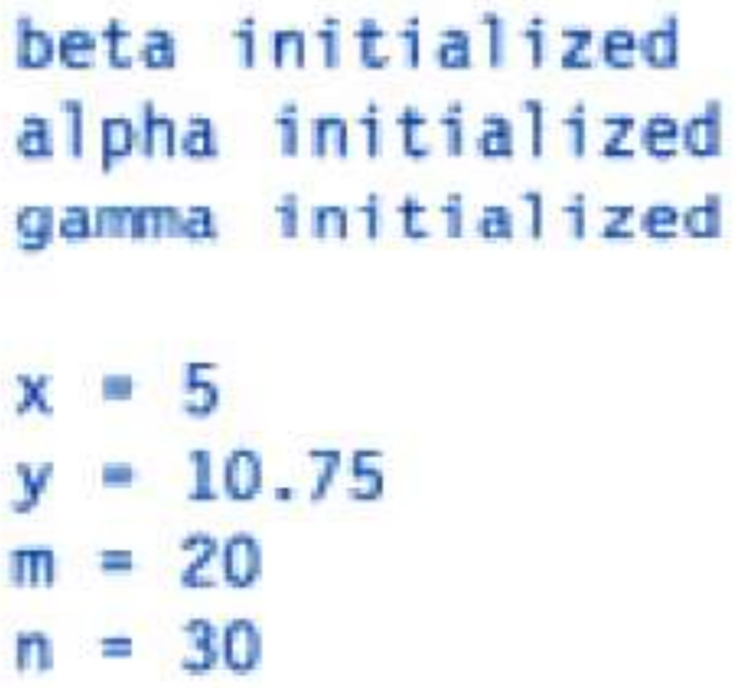
```
class alpha
{
      int x;
   public:
      alpha(int i)
      {
            x = i;
            cout << "alpha initialized \n";
      }
      void show_x(void)
      { cout << "x = " << x << "\n"; }
};
```

```
class beta
{
    float y;
  public:
    beta(float j)
    {
        y = j;
        cout << "beta initialized \n";
    }
    void show_y(void)
    { cout << "y = " << y << "\n"; }
};
```

```
class gamma: public beta, public alpha
{
        int m, n;
    public:
        gamma(int a, float b, int c, int d):
                alpha(a), beta(b)
        {
                m = c;
                n = d;
                cout << "gamma initialized \n";
        }
    void show_mn(void)
    {
        cout << "m = " << m << "\n"
             << "n = " << n << "\n";
    }
};

int main()
{
    gamma g(5, 10.75, 20, 30);
    cout << "\n";
    g.show_x();
    g.show_y();
    g.show_mn();

    return 0;
}
```



```
beta initialized
alpha initialized
gamma initialized

x = 5
y = 10.75
m = 20
n = 30
```

- **Points to Remember**

1. Whenever the derived class's default constructor is called, the base class's default constructor is called automatically.
2. To call the parameterized constructor of base class inside the parameterized constructor of sub class, we have to mention it explicitly.
3. The parameterized constructor of base class cannot be called in default constructor of sub class, it should be called in the parameterized constructor of sub class.

References:

- www.studytonight.com
- www.tutorialpoint.com
- www.geeksforgeeks.org
- “Object oriented programming in C++” Robert Lafore
- “Object oriented programming with C++”, E.Balagurusamy