UNITIII

OBJECT-ORIENTED PROGRAMMING CONCEPTS

- **Topics to be discussed,**

➢Inheritance

➢**Constructors and Destructors in Derived Classes**

➢Polymorphism and Virtual Functions

Constructors and Destructors in Derived Classes

- 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.

Constructors in Derived Class Example

```
#include <iostream>
using namespace std;
class Base
{
protected:
   Base()
   {
      cout<<"\nBase class Constructor";
   }
};
class Derived: public Base
{
public:
   Derived()
   {
      cout<<"\nDerived class Constructor";
   }
};

int main()
{
   Derived d;
   return 0;
}
```

Output:

```
Base class Constructor
Derived class Constructor
```

Constructors and Destructors in Derived Classes – Cont'd

- **Why the base class's constructor is called on creating an object of derived class?**
  - 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 specifier 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.**

Constructors and Destructors in Derived Classes – Cont'd

- **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.
  - for example if we have defined like this "class Derived: public A, public B", then Constructor of class A will be called, then constructor of class B will be called.

Order of constructor call for Multiple Inheritance - Example

```
#include <iostream>
using namespace std;
class A
{
public:
   A()
   {
      cout<<"Class A Constructor\n";
   }
};
class B
{
public:
   B()
   {
      cout<<"Class B Constructor\n";
   }
};
class Derived: public A, public B
{
public:
   Derived()
   {
      cout<<"Derived class Constructor\n";
   }
};
int main()
{
   Derived d;
   return 0;
}
```

Output:

```
Class A Constructor
Class B Constructor
Derived class Constructor
```

Constructors and Destructors in Derived Classes – Cont'd

- **Inheritance in Parameterized Constructor**
  - In the case of the default constructor, it is implicitly accessible from parent to the child class but parameterized constructors are not accessible to the derived class automatically, for this reason, an explicit call has to be made in the child class constructor to access the parameterized constructor of the parent class
- **Syntax Example:**

Derived-Constructor (arg1, arg2, arg3….): Base 1-Constructor (arg1,arg2), Base 2-Constructor(arg3,arg4)

{ …. }

```cpp
#include <iostream>
using namespace std;
class Base
{
protected:
   int x;
public:
   Base(int x)
   {
      this->x=x;
      cout<<"\nBase class Constructor,x="<<x;
   }
};
class Derived: public Base
{
   int y;
public:
   Derived()
   {
        cout<<"\nDerived class Constructor";
         cout<<"\nx="<<x<<"\ny="<<y;
   }
};
int main()
{
   Derived d;
   return 0;
}
```

File	Line	Message
		=== Build file: "no target" in "no project" (compiler: unknown) ===
H:\2024\CS320...		In constructor 'Derived::Derived()':
H:\2024\CS320...	19	error: no matching function for call to 'Base::Base()'
H:\2024\CS320...	8	note: candidate: 'Base::Base(int)'
H:\2024\CS320...	8	note: candidate expects 1 argument, 0 provided
H:\2024\CS320...	3	note: candidate: 'constexpr Base::Base(const Base&)'
H:\2024\CS320...	3	note: candidate expects 1 argument, 0 provided
H:\2024\CS320...	3	note: candidate: 'constexpr Base::Base(Base&&)'
H:\2024\CS320...	3	note: candidate expects 1 argument, 0 provided
		=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) =

Constructors and Destructors in Derived Classes – Cont'd

- **Important Points**:
  - Whenever the derived class's default constructor is called, the base class's default constructor is called automatically.
  - To call the parameterized constructor of base class inside the parameterized constructor of sub class, we have to mention it explicitly.

Inheritance in Parameterized Constructor - Example

```
#include <iostream>
using namespace std;
class Base
{
   protected:
       int x;
   public:
      Base (int k)   //parameterized constructor of Base class.
     {
        cout<<"\nBase class Parameterized Constructor";
        x = k;
     }
};
class Derived: public Base
{
  int y;
  public:
  Derived(int a, int b):Base(a)  //constructor of child class calling constructor of base class.
  {
     cout<<"\nDerived class Parameterized Constructor";
     y = b;
  }
  void display()
  {
    cout<<"\nx="<<x;
    cout<<"\ny="<<y;
  }
};
```

```
int main()
{
  Derived obj(2,3);
  obj.display();
}
```

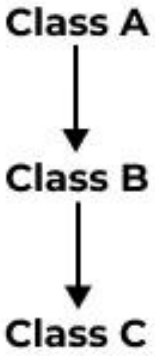
Output:

```
Base class Parameterized Constructor
Derived class Parameterized Constructor
x=2
y=3
```

Constructors and Destructors in Derived Classes – Cont'd

- Destructors in C++ are called in the opposite order of that of Constructors.
- In inheritance, the order of constructors calling is: from *child* class to *parent* class (*child* **->** *parent*).
- In inheritance, the order of constructors execution is: from *parent* class to *child* class (*parent* **->** *class*).
- In inheritance, the order of destructors calling is: from *child* class to *parent* class (*child* **->** *parent*).
- In inheritance, the order of destructors execution is: from *child* class to *parent* class (*child* **->** *parent*).

Order of Calling For Constructors & Destructors in Inheritance



Order of Constructor Call

A() - Class A Constructor

B() - Class B Constructor

C() - Class C Constructor

Order of Destructor Call

C() - Class C Destructor

B() - Class B Destructor

A() - Class A Destructor

Destructors in Derived Classes - Example

```
#include<iostream>
using namespace std;
class baseClass
{
public:
 baseClass()
 {
  cout <<"\nI am baseClass constructor";
 }
  ~baseClass()
 {
  cout <<"\nI am baseClass destructor";
 }
};
class derivedClass: public baseClass
{
public:
 derivedClass()
 {
  cout <<"\nI am derivedClass constructor";
 }
 ~derivedClass()
 {
  cout <<"\nI am derivedClass destructor";
 }
};
```

```
int main()
{
derivedClass D;
return 0;
}
```

Output:

```
I am baseClass constructor
I am derivedClass constructor
I am derivedClass destructor
I am baseClass destructor
```

Constructors and Destructors in Derived Classes – Cont'd

- **Constructor & Destructor in Multiple inheritance**

```
class C: public A, public B
{
        //...
};
```

  - Here, A class in inherited first, so constructor of class A is called first then the constructor of class B will be called next.
  - The destructor of derived class will be called first then destructor of base class which is mentioned in the derived class declaration is called from last towards first sequence wise.

Constructor & Destructor in Multiple inheritance

```
#include<iostream>
using namespace std;
class baseClass1
{
 public:
  baseClass1()
  {
   cout<<"\nI am baseClass1 constructor";
  }
  ~baseClass1()
  {
   cout<<"\nI am baseClass1 destructor";
  }
};
class baseClass2
{
public:
 baseClass2()
 {
  cout<<"\nI am baseClass2 constructor";
 }
 ~baseClass2()
 {
  cout<<"\nI am baseClass2 destructor";
 }
};
class derivedClass: public baseClass1, public baseClass2
{
  public:
   derivedClass()
   {
    cout<<"\nI am derivedClass constructor";
   }
   ~derivedClass()
   {
    cout<<"\nI am derivedClass destructor";
   }
};
 int main()
 {
 derivedClass D;
 return 0;
}
```

Output:

```
I am baseClass1 constructor
I am baseClass2 constructor
I am derivedClass constructor
I am derivedClass destructor
I am baseClass2 destructor
I am baseClass1 destructor
```