C++

Constructor

INHERITANCE IN C++

By – Mohammad Imran

Inheritance with Constructor

Constructor is a class member function with the same name as the class name. The main job of the constructor is to allocate memory for class objects. Constructor is automatically called when the object is created. It is very important to understand how constructors are called in inheritance.

We know when we create an object, then automatically the constructor of the class is called and that constructor takes the responsibility to create and initialize the class members. Once we create the object, then we can access the data members and member functions of the class using the object.

By – Mohammad Imran

Inheritance with Constructor

In inheritance, we have Base/Parent/Superclass as well as Derived/Child/Subclass. If we create an object of the Base class, then the base class constructor is called and initializes the base class members, and then using the base class object, we can call the data members and member functions of the base class.

When we create an instance of the Derived class, then the Derived Class constructor is called and initializes the Derived Class members. But using the Derived class object we can access both the base class and derived class members. How?

By – Mohammad Imran

Inheritance with Constructor

Example - 1

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

OUTPUT

```
Base Constructor
Derived Constructor
```

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

By – Mohammad Imran

Inheritance with Constructor

Example - 2

```
#include <iostream>
using namespace std;
class Base
{
  public:
   Base ()
   {
      cout << "Base Constructor" << endl;
   }
   Base (int x)
   {
      cout << "Base Para Constructor:" << x << endl;
   }
};
```

By – Mohammad Imran

Inheritance with Constructor

Example - 2

```
class Derived : public Base
{
  public:
   Derived()
   {
      cout << "Derived Constructor" << endl;
   }
   Derived (int a)
   {
      cout << "Base Para Constructor:" << x <<
      endl;
   }
};
```

OUTPUT

```
Base Constructor
Derived Para Constructor: 5
```

```
int main()
{
   Derived d(5);
   return 0;
}
```

By – Mohammad Imran

Inheritance with Constructor

Calling Parameterize Constructor of Base Class

Now we want to call the parameterized constructor of the Base class when the object of the Derived classes is executed. So, for that, we should have a special constructor in the Derived class as follows which will call the base class parameterized constructor.

Inheritance with Constructor

Example - 3

Calling Parameterize Constructor of Base Class

```
#include <iostream>
using namespace std;
class Base
{
  public:
   Base ()
   {
      cout << "Base Constructor" << endl;
   }
   Base (int x)
   {
      cout << "Base Para Constructor:" << x << endl;
   }
};
```

By – Mohammad Imran

Inheritance with Constructor

Example - 3

```
class Derived : public Base
{
  public:
   Derived()
   {
      cout << "Derived Constructor" << endl;
   }
   Derived (int a):Base (a)
   {
      cout << "Base Para Constructor:" << x <<
      endl;
   }
};
```

OUTPUT

```
Base Para Constructor: 5
Derived Para Constructor: 5
```

```
int main()
{
   Derived d(5);
   return 0;
}
```

By – Mohammad Imran

Function Overloading

Introduction

Function overloading is a feature in object-oriented programming (OOP) that allows multiple functions with the same name to be defined, as long as they have different parameter lists. This means that a class can have multiple functions with the same name, but with different numbers or types of parameters.

By – Mohammad Imran

Function Overloading

Example - 4

```
#include <iostream>
using namespace std;
class Calculator
{
  public:
   int add(int a, int b)
   {
      return a + b;
   }
   double add(double a, double b)
   {
      return a + b;
   }
   int add(int a, int b, int c)
   {
      return a + b + c;
   }
};
```

```
int main()
{
   Calculator cal;
   cout << cal.add(10,20,30;
   return 0;
}
```

OUTPUT

60

By – Mohammad Imran

Function Overloading

How it Works

When a function is called, the compiler checks the number and types of parameters passed to the function and matches it with the function signature that best fits. This is known as function resolution.

In the example above, if we create an instance of the Calculator class and call the add function with two int parameters, the compiler will call the **`add(int, int)`** function. If we call the add function with two double parameters, the compiler will call the **`add(double, double)`** function. And if we call the add function with three int parameters, the compiler will call the **`add(int, int, int)`** function.

Function Overloading

Benefits

- ✓ **Improved Code Readability**: By using the same function name for similar operations, the code becomes more readable and easier to understand.
- ✓ **Increased Flexibility**: Function overloading allows for more flexibility in how functions can be called, making it easier to write code that is more expressive and concise.
- ✓ **Reduced Code Duplication**: By defining multiple functions with the same name, we can avoid duplicating code and reduce the overall size of the program.

Function Overriding

Introduction

Function overriding in C++ is a concept by which you can define a function of the same name and the same function signature (parameters and their data types) in both the base class and derived class with a different function definition. It redefines a function of the base class inside the derived class, which overrides the base class function. Function overriding is an implementation of the run-time polymorphism. So, it overrides the function at the run-time of the program.

By – Mohammad Imran

Function Overriding

Benefits

- ✓ Function overriding allows for dynamic dispatch, enabling more flexible and reusable code.
- ✓ It helps implement polymorphic behavior, allowing different objects to be treated as instances of their base type while still invoking their specific implementations.

Function Overriding

Syntax

```
class Base
{
  public:
   returnType functionName(parameters)
   {
      // Base class implementation
   }
};
class Derived : public Base
{
  public:
   returnType functionName(parameters)
   {
      // Derived class implementation
   }
};
```

By – Mohammad Imran

Function Overriding

Example - 5

```
#include <iostream>
using namespace std;
class Base
{
  public:
   void input(int a, int b)
   {
      cout << a << " and " << b << endl;
   }
};
```

By – Mohammad Imran

Function Overriding

Example - 5

```
class Derived : public Base
{
   int n1, n2, res;
  public:
   void input(int a, int b)
   {
      n1 = a;
      n2 = b;
      res = n1 + n2;
   }
   void show()
   {
      cout << "Sum of input = " << res;
   }
};
```

OUTPUT

```
Sum of input = 150
```

```
int main()
{
      Derived d;
      d.input(100,50);
      d.show();
      return 0;
}
```

By – Mohammad Imran

Function Overriding

Pointer Object of Base Class

Example - 6

```
#include <iostream>
using namespace std;
class Animal
{
  public:
   void makeSound()
   {
      cout << "Animal makes generic sound." << endl;
   }
};
```

By – Mohammad Imran

Function Overriding

Pointer Object of Base Class

Example - 6

```
class Cat : public Animal
{
  public:
   void makeSound()
   {
      cout << "Cat meows." << endl;
   }
};
int main()
{
      Animal *animalPtr;
      Cat cat;
      animalPtr = &cat;
      animalPtr->makeSound();
      return 0;
}
```

OUTPUT

```
Animal makes generic sound.
```

By – Mohammad Imran

Coding Questions

By – Mohammad Imran

Question - 1

In a C++ program designed to perform basic arithmetic operations, you have two classes: **Addition** and **Multiplication**. The **Addition** class takes two integers by argument constructor and computes their sum, while the **Multiplication** class inherits from **Addition** and computes the product of the same two integers using argument constructor while we created only derived class object.

Click here to see code

By – Mohammad Imran

QUIZ

By – Mohammad Imran

Quiz - 1

```
class A
{
  public:
   void foo()
   {
      cout << "A::foo()" << endl;
   }
};
class B : public A
{
  public:
   void foo()
   {
      cout << "B::foo()" << endl;
   }
};
```

```
class C : public B
{
  public:
   void bar()
   {
      foo();
   }
};
int main()
{
   C obj;
   obj.bar();
   return 0;
}
```

OUTPUT

```
B::foo()
```

By – Mohammad Imran