Constructor and Destructor in C++

By – Mohammad Imran

Constructor Introduction

Constructor is a special type of method in a class that has the same name as class name without any return type in public scope.

Constructor is basically used to initialize member variables for an object automatically by default value or value passed by the user when object of class is created.

Constructor is automatically called whenever object of class is created.

By – Mohammad Imran

Constructor

Key Points

- ✓ Constructor's name is the same as the class name.
- ✓ There is no return type for constructors.
- ✓ An object's constructor is invoked automatically during object creation.
- ✓ The C++ compiler creates a default constructor for the object if a constructor is not specified (expects any parameters and has an empty body).

By – Mohammad Imran

Constructor Characteristics

- ✓ A constructor can be made public, private, or protected per our program's design. Constructors are mostly made public, as public methods are accessible from everywhere, thus allowing us to create the object of the class anywhere in the code. When a constructor is made private, other classes cannot create instances of the class. This is used when there is no need for object creation. Such a case arises when the class only contains static member.

Constructor

Characteristics

- ✓ A constructor in C++ cannot be inherited. However, a derived class can call the base class constructor. A derived class(i.e., child class) contains all members and member functions(including constructors) of the base class.
- ✓ Constructor functions are not inherited, and their addresses cannot be referenced.

By – Mohammad Imran

Constructor

Syntax

```
class <class_name>
{
   <access_specifier>:
    <class_name>() //Constructor
    {
        Statements;
    }
};
```

By – Mohammad Imran

Types of Constructors in C++

01 **Default Constructor**

02 **Parameterized Constructor**

03 **Copy Constructor**

By – Mohammad Imran

Constructor Types

Basically, there are three types of constructor

- ✓ Default Constructor
- ✓ Parameterized Constructor
- ✓ Copy Constructor

Default Constructor

Default constructor is the constructor which doesn’t take any argument. It has no parameters. It is also called a zero-argument constructor.

The default constructor is used to initialize all the class elements at the time of object creation.

The default constructor is one of the most important constructors. If we do not explicitly create any type of constructor in C++, then the compiler automatically creates a default constructor for the particular object.

Default Constructor

Example - 1

```
#include <iostream>
using namespace std;
class Construct
{
  public:
   int a, b;

   //Default Constructor
   Construct()
   {
      a = 10;
      b = 20;
   }
};
```

```
int main()
{
   Construct c;
   cout << "a: " << c.a << endl;
   cout << "b: " << c.b << endl;
   return 0;
}
```

OUTPUT

```
a: 10
b: 20
```

By – Mohammad Imran

Constructor

Parameterize Constructor

Like argument function we can create an argument constructor known as parameterize constructor

Using the default constructor, it is impossible to initialize different objects with different initial values. What if we need to pass arguments to constructors which are needed to initialize an object? There is a different type of constructor called Parameterized Constructor to solve this issue.

A Parameterized Constructor is a constructor that can accept one or more arguments. This helps programmers assign varied initial values to an object at the creation time

Parameterize Constructor

Example - 2

```
#include <iostream>
using namespace std;
class Constructor
{
  public:
   int a, b;
   Constructor(int x, int y)
   {
      a = x;
      b = y;
   }
   void show()
   {
      cout << "a = " << a << endl;
      cout << "b = " << b << endl;
   }
};
```

```
int main()
{
   Constructor c(200, 100);
   c.show();
   return 0;
}
```

OUTPUT

```
a = 200
b = 100
```

By – Mohammad Imran

Parameterize Constructor

Example - 3

```
#include<iostream>
#include<string.h>
using namespace std;
class Student
{
   int rno;
   char name[50];
   double fee;
  public:
   Student(int no,char n[])
   {
      rno = no;
      strcpy(name,n);
   }
   void display()
   {
      cout << rno << endl;
      cout << name << endl;
   }
};
int main()
{
   Student s(1001,"David");
   s.display();
   return 0;
}
```

OUTPUT

```
1001
David
```

By – Mohammad Imran

Constructor

Copy Constructor

As the name suggests, a copy constructor is a constructor that initializes an object using another object of the same class. We usually use the copy constructor to copy data from one object to another.

Copy constructor takes a reference to an object of the same class as an argument.

A Copy constructor in C++ is a type of constructor used to create a copy of an already existing object of a class type. The compiler provides a default Copy Constructor to all the classes.

Copy Constructor

Syntax

```
class <class_name>
{
   <class_name>(Class &Object)    //Copy Constructor
   {
      Statements;
   }
};
```

By – Mohammad Imran

Copy Constructor

Example - 4

```
#include <iostream>
#include <string.h>
using namespace std;
class Teacher
{
    int tId;
    string tName;
    double tSalary;
   public:
    Teacher(int no, string n, double f)
    {
        tId = no;
        tName = n;
        tSalary = f;
    }
```

By – Mohammad Imran

Copy Constructor

Example - 4

```
   Teacher(Teacher& t)
   {
      tId = t.tId;
      tName = t.tName;
      tSalary = t.tSalary;
      cout << "Copy Constructor Called" << endl;
   }
   void display();
};

void Teacher::display()
{
    cout << tId << "\t" << tName << "\t" << tSalary << endl;
}
```

By – Mohammad Imran

Copy Constructor

Example - 4

```
int main()
{
   Teacher s(1001, "Manjeet", 10000);
   s.display();
   Teacher t1(s);
   t1.display();
   return 0;
}
```

OUTPUT

```
1001 Manjeet 10000
Copy Constructor Called
1001 Manjeet 10000
```

By – Mohammad Imran

Copy Constructor

When Called

In C++, a copy constructor may be called in the following cases:

- ✓ When an object of the class is returned by value.
- ✓ When an object of the class is passed (to a function) by value as an argument.
- ✓ When an object is constructed based on another object of the same class.
- ✓ When the compiler generates a temporary object.

Copy Constructor

Types

If we don’t define our own copy constructor, the C++ compiler creates a default copy constructor for each class which works fine in general. However, **we need to define our own copy constructor only if an object has pointers or any runtime allocation** of the resource like *a file handle*, a network connection, etc. because the default ***constructor does only shallow copy.***

- ✓ Shallow Copy
- ✓ Deep Copy

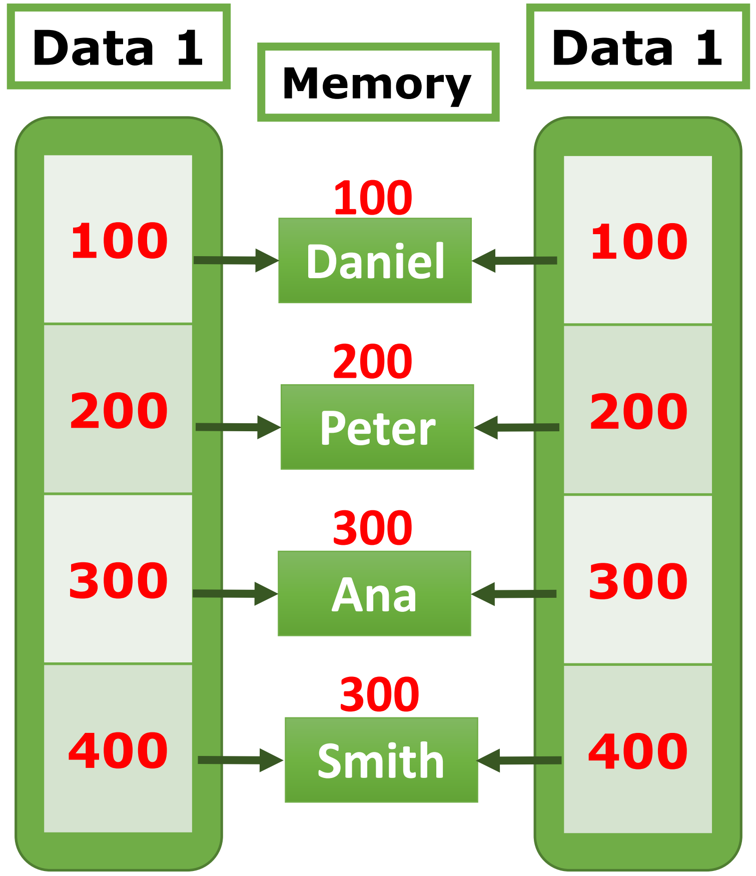
By – Mohammad Imran

Copy Constructor

Shallow Copy

Shallow Copy means that only the pointers will be copied not the actual resources that the pointers are pointing to. This can lead to dangling pointers if the original object is deleted.

- ✓ The default copy constructor can only produce the shallow copy.
- ✓ A Shallow copy is defined as the process of creating the copy of an object by copying data of all the member variables as it is.



By – Mohammad Imran

Copy Constructor | Shallow Copy | Example - 5

```
#include <iostream>
using namespace std;
class ShallowCopy
{
    int a, b, *p;
   public:
    ShallowCopy()
    {
       p = new int;
    }
    void setdata(int x,int y,int z)
    {
       a=x;
       b=y;
       *p=z;
    }
    void showdata()
    {
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "*p = " << *p << endl;
    }
};
```

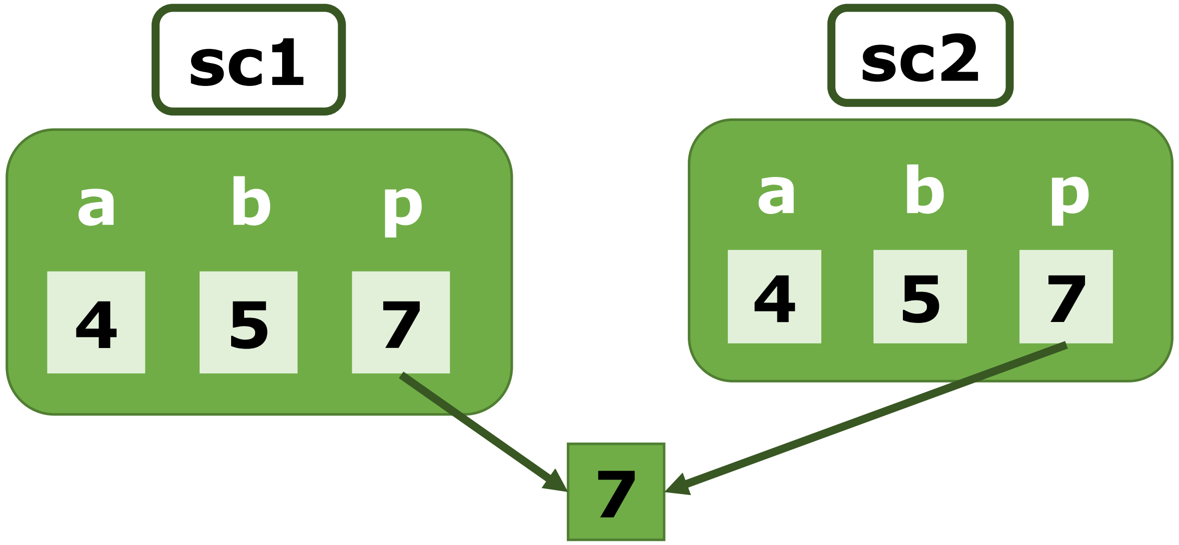
By – Mohammad Imran

Copy Constructor | Shallow Copy | Example - 5

```
int main()
{
    ShallowCopy sc1;
    sc1.setdata(4, 5, 7);
    sc1.showdata();
    ShallowCopy sc2 = sc1;
    sc2.setdata(5, 10, 15);
    sc2.showdata();
    sc1.showdata();
}
```

OUTPUT

```
 a = 4      a = 5      a = 4
 b = 5      b = 10     b = 4
*p = 7     *p = 15    *p = 15
```



By – Mohammad Imran

Copy Constructor

Shallow Copy **Description about Code**

In the above case, a programmer has not defined any constructor, therefore, the statement **Demo d2 = d1;** calls the default constructor defined by the compiler. The default constructor creates the exact copy or shallow copy of the existing object. Thus, the pointer p of both the objects point to the same memory location. Therefore, when the memory of a field is freed, the memory of another field is also automatically freed as both the fields point to the same memory location. This problem is solved by the **user-defined constructor** that creates the **Deep copy**.

By – Mohammad Imran

Copy Constructor

Deep Copy

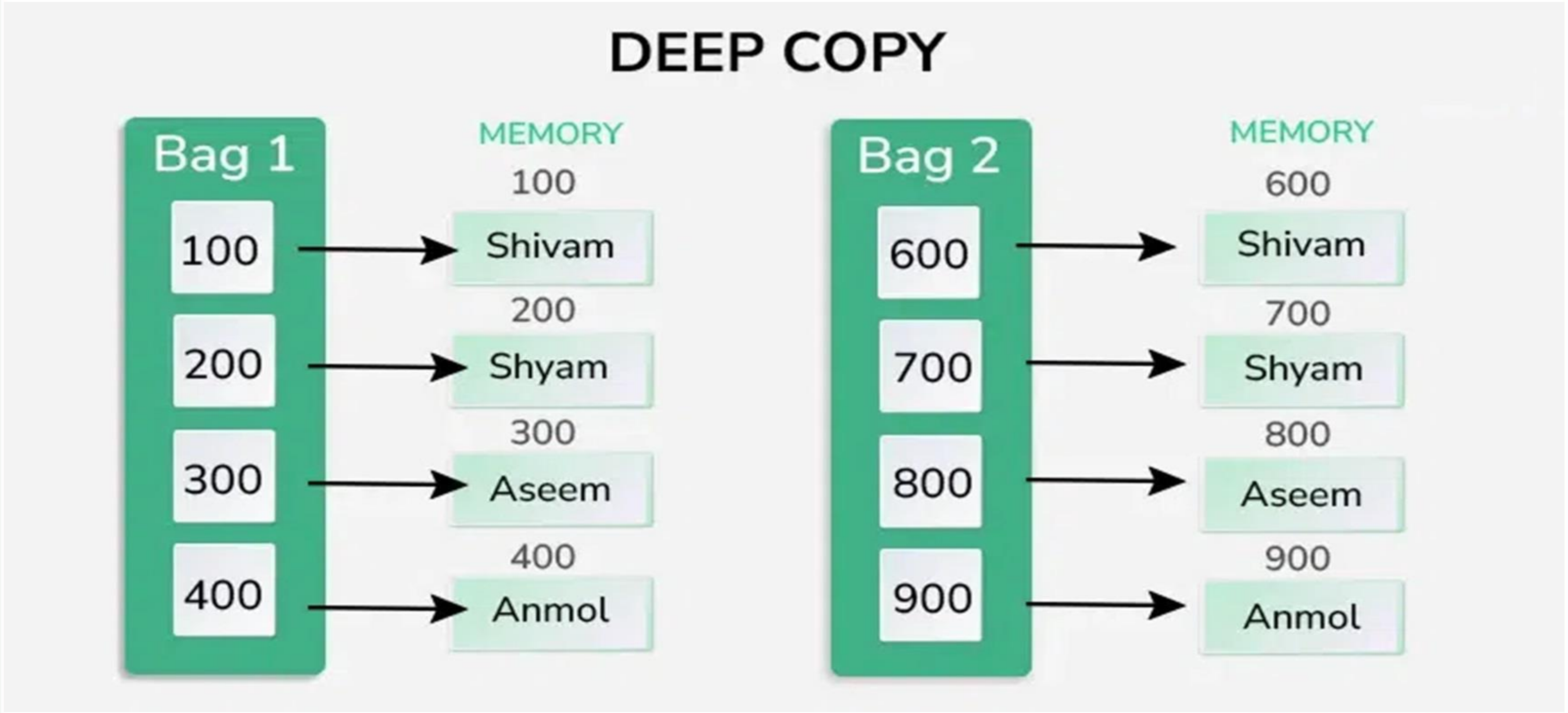
Deep copy dynamically allocates the memory for the copy and then copies the actual value, both the source and copy have distinct memory locations. In this way, both the source and copy are distinct and will not share the same memory location. Deep copy requires us to write the user-defined constructor.

Deep copy is possible only with a user-defined copy constructor. In a user-defined copy constructor, we make sure that pointers (or references) of copied objects point to new copy of the dynamic resource allocated manually in the copy constructor using new operators.

By – Mohammad Imran

Copy Constructor

Deep Copy



By – Mohammad Imran

Copy Constructor | Deep Copy | Example - 6

```
#include <iostream>
using namespace std;
class DeepCopy
{
   int a, b, *p;
  public:
   DeepCopy()
   {
      p = new int;
   }
   DeepCopy(DeepCopy &dc)
   {
      a = dc.a;
      b = dc.b;
      p = new int;
      *p = *(dc.p);
   }
```

```
   void showdata()
   {
      cout << "a = " << a << endl;
      cout << "b = " << b << endl;
      cout << "*p = " << *p << endl;
   }

   void setdata(int x,int y,int z)
   {
      a = x;
      b = y;
      *p = z;
   }
};
```

By – Mohammad Imran

Copy Constructor | Deep Copy | Example - 6

```
int main()
{
   DeepCopy dc1;
   dc1.setdata(25, 50, 75);
   dc1.showdata();
   DeepCopy dc2 = dc1;
   dc2.setdata(5, 10, 15);
   dc2.showdata();
   dc1.showdata();
}
```

OUTPUT

```
 a = 25      a = 5       a = 25
 b = 50      b = 10      b = 50
*p = 75     *p = 15     *p = 75
```

By – Mohammad Imran

Copy Constructor | Deep Copy | Example - 7

```
#include <cstring>
#include <iostream>
using namespace std;
class String
{
  private:
   char* s;
   int size;
  public:
   String(char* str = NULL);
   ~String()
   {
      delete[] s;
   }
   String(String&);
```

```
   void print()
   {
      cout << s << endl;
   }
   void change(char*);
};

String::String(char* str)
{
   size = strlen(str);
   s = new char[size + 1];
   strcpy(s, str);
}
```

By – Mohammad Imran

Copy Constructor | Deep Copy | Example - 7

```
void String::change(char* str)
{
   delete[] s;
   size = strlen(str);
   s = new char[size + 1];
   strcpy(s, str);
}

String::String(String& oldstr)
{
   size = oldstr.size;
   s = new char[size + 1];
   strcpy(s, oldstr.s);
}
```

```
int main()
{
     String str1("Hello");
     String str2 = str1;

     str1.print();
     str2.print();

     str2.change("Hello World");

     str1.print();
     str2.print();
     return 0;
}
```

By – Mohammad Imran

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.

Destructor

Key Points

- ✓ Destructor is invoked automatically by the compiler when its corresponding constructor goes out of scope and releases the memory space that is no longer required by the program.
- ✓ Destructor neither requires any argument nor returns any value therefore it cannot be overloaded.
- ✓ Destructor cannot be declared as static and const;
- ✓ Destructor should be declared in the public section of the program.
- ✓ Destructor is called in the reverse order of its constructor invocation.

By – Mohammad Imran

Destructor

Syntax

```
class <class_name>
{
   ~ <class_name>()     //Destructor
   {
      Statements;
   }
};
```

By – Mohammad Imran

Destructor

Example - 7

```
#include <iostream>
using namespace std;
class Test
{
   public:
    Test()
    {
       cout << "Constructor Called\n";
    }
    ~Test()
    {
       cout << "Destructor Called";
    }
};
```

OUTPUT

```
Constructor Called
Destructor Called
```

```
int main()
{
     Test t;
     return 0;
}
```

By – Mohammad Imran

Destructor

Example - 8

```
#include <iostream>
using namespace std;
class Department
{
   public:
    Department()
    {
        cout << "Department Class Constructor" << endl;
    }
    ~Department()
    {
        cout << "Department Class Destructor" << endl;
    }
};
```

By – Mohammad Imran

Destructor

Example - 9

```
class Employee
{
   public:
    Employee()
    {
       cout << "Employee Class Constructor" << endl;
    }
    ~Employee()
    {
       cout << "Employee Class Destructor" << endl;
    }
};
```

By – Mohammad Imran

Destructor

Example - 9

```
int main(void)
{
   Department d1;
   Employee e2;
   return 0;
}
```

OUTPUT

```
Department Class Constructor
Employee Class Constructor
Employee Class Destructor
Department Class Destructor
```

By – Mohammad Imran

Constructor and Destructor Calling Order

Example - 10

```
#include <iostream>
using namespace std;
class Test
{
  public:
   Test()
   {
      cout << "Constructor Called\n";
   }
   ~Test()
   {
      cout << "Destructor Called";
   }
};
```

OUTPUT

```
Constructor Called
Constructor Called
Destructor Called
Destructor Called
```

```
int main()
{
     Test t1, t2;
     return 0;
}
```

By – Mohammad Imran

Constructor Overloading

Syntax

```
class <class_name>
{
   <class_name>()
   {
      Statements;
   }
   <class_name>(int)
   {
      Statements;
   }
   <class_name>(int, float)
   {
      Statements;
   }
};
```

By – Mohammad Imran

Constructor Overloading

Example - 11

```
#include<iostream>
using namespace std;
class Cons
{
  public:
   Cons()
   {
      cout << "Default Constructor Called\n";
   }
   Cons(int x)
   {
      cout << "Argument Constructor Called\n";
      cout << "Value of x = " << x << endl;
   }
```

By – Mohammad Imran

Constructor Overloading

Example - 11

```
    Cons(int x, int y)
    {
       cout << "Sum = "<< x+y << endl;
    }
};

int main()
{
    Cons c1;
    Cons c2(101);
    Cons c3(50,20);
    return 0;
}
```

OUTPUT

```
Default Constructor Called
Argument Constructor Called
Value of x = 101
Sum = 70
```

By – Mohammad Imran

Dynamic Constructor

When allocation of memory is done dynamically using dynamic memory allocator **new** in a constructor, it is known as **dynamic constructor**. By using this, we can dynamically initialize the objects.

The constructor which allocates a block of memory that can be accessed by the objects at run time is known as **Dynamic Constructor**.

Dynamic Constructor

Example - 12

```
#include<iostream>
using namespace std;
class Dynamic
{
   const char *ch;
  public:
   Dynamic()
   {
      ch = new char[6];
      ch = "University";
   }
   void display()
   {
      cout << ch << endl;
   }
   ~Dynamic()
   {
      delete ch;
   }
};
int main()
{
   Dynamic d;
   d.display();
   return 0;
}
```

OUTPUT

```
University
```

By – Mohammad Imran

Constructor and Destructor

With Static Member

Static members are often used in constructors and destructors to manage and track resources or states that affect all instances of a class. Here’s a detailed breakdown of how static members interact with constructors and destructors:

Usage:

Tracking Instance Count: A static member variable is commonly used to keep track of how many objects of a class have been created.

Constructor and Destructor

With Static Member

Static Member Variable

Tracking Instance Count: A static member variable is commonly used to keep track of how many objects of a class have been created.

Constructor and Destructor

With Static Member

Constructor

The constructor of a class is invoked when a new object is instantiated. Within the constructor, you can modify static member variables to keep track of changes that affect all instances.

Example Behavior:

Incrementing a Count: Each time an object is created, you might want to increment a static counter to reflect the total number of active objects.

Constructor and Destructor

With Static Member

Destructor

The destructor is called when an object is destroyed. It provides a way to clean up resources or update class-wide information when an object is removed.

Example Behavior:

Decrementing a Count: When an object is destroyed, you might want to decrement a static counter to keep an accurate count of active objects.

Constructor and Destructor | With Static Member | Example - 13

```
#include <iostream>
using namespace std;
class MyClass
{
  private:
   static int objCnt;
  public:
   MyClass() // Constructor
   {
      ++objCnt;
      cout << "Object created and count = " << objCnt << endl;
   }
   ~MyClass()
   {
      -- objCnt; // Decrement when object is destroyed
      cout << "Object destroyed and count = " << objCnt << endl;
   }
```

By – Mohammad Imran

Constructor and Destructor
With Static Member **Example - 13**

```
    // Static member function to get the current count of instances
    static int getInstanceCount()
    {
        return instanceCount;
    }
};

int MyClass::objCnt = 0;
```

By – Mohammad Imran

Constructor and Destructor

With Static Member **Example - 13**

```
int main()
{
   cout << "Initial instance count: " << MyClass::getInstanceCount() << endl;
   MyClass obj1;
   MyClass obj2;
   MyClass obj3;
   cout << "Instance count after creating objects: " << MyClass::getInstanceCount() << endl;
   // Destroy objects
   {
      MyClass obj4;
      cout << "Instance count after creating one more object: " << MyClass::getInstanceCount() << endl;
   }
```

Constructor and Destructor

With Static Member **Example - 13**

```
    // Display the count of instances after an object is destroyed

    cout << "Instance count after destroying an object: " <<
    MyClass::getInstanceCount() << endl;
    return 0;
}
```

By – Mohammad Imran

Constructor and Destructor

With Static Member **Example - 13**

OUTPUT

```
Initial instance count: 0
Object created and count = 1
Object created and count = 2
Object created and count = 3
Instance count after creating objects: 3
Object created and count = 4
Instance count after creating one more object: 4
Object destroyed and count = 3
Instance count after destroying an object: 3
Object destroyed and count = 2
Object destroyed and count = 1
Object destroyed and count = 0
```

By – Mohammad Imran

Coding Questions

By – Mohammad Imran

Question - 1

Write a C++ program to demonstrate the encapsulation feature of OOPs.

Click here to see code

By – Mohammad Imran

MyAnatomy

QUIZ

By – Mohammad Imran

Quiz - 1

MyAnatomy

What is the primary purpose of a constructor in C++?

A. To create a copy of an object

B. To initialize an object of a class

C. To free up memory

D. To define methods for a class

ANSWER

B

By – Mohammad Imran

Quiz - 2

MyAnatomy

Which of the following statements about destructors is true?

A. A destructor has the same name as the class but with an additional underscore

B. A destructor can be overloaded.

C. Destructors are automatically called when an object goes out of scope or is explicitly deleted.

D. Destructors can take parameters.

<u>**ANSWER**</u>

B

By – Mohammad Imran

Quiz - 3

MyAnatomy

How do you declare a default constructor in a class?

A. `MyClass() = default;`

B. `MyClass() {};`

C. `MyClass();`

D. `MyClass() {}`

ANSWER

D

By – Mohammad Imran

Quiz - 4

MyAnatomy

What happens if a class does not have a user-defined constructor?

A. The compiler will generate a default constructor.

B. The program will not compile.

C. The class will be unable to create objects.

D. The class will only have a destructor.

ANSWER

A

By – Mohammad Imran

Quiz - 5

MyAnatomy

Which of the following is the correct way to call a destructor explicitly in C++?

A. `delete objectName;`

B. `objectName.~ClassName();`

C. `objectName.destructor();`

D. `objectName.destruct();`

ANSWER

B

By – Mohammad Imran

Quiz - 5

MyAnatomy

What is the purpose of a copy constructor?

A. To initialize a class's static members.

B. To delete an existing object.

C. To create a new object with the same state as an existing object.

D. To handle file operations

ANSWER

C

By – Mohammad Imran

Quiz - 6

MyAnatomy

What will be output of the code?

```
class A
{
   int a;
     public:
   A(int i)
   {
      a = i;
   }
   void assign(int i)
   {
      a = i;
   }
   int return_value()
   {
      return a;
   }
};
int main(int argc, char const *argv[])
{
   A obj;
   obj.assign(5);
   cout << obj.return_value();
   return 0;
}
```

OUTPUT

Error

By – Mohammad Imran