Lecture 13-14
Constructors

Contents

Introduction to Constructor

- A constructor is a special member function of a class whose task is to initialize the objects of its class.
- It has the same name as of that class.
- It is invoked on the creation of the object of the associated class.
- It can be defined either inside the class or outside the class.

Characteristics of Constructor

- It must be declared in the public section of the class.
- It does not have any return type (not even void) and therefore cannot return any value.
- It is automatically called when the object is created.
- It can also be called explicitly.
- It can take parameters.
- Constructors can have default arguments.
- Constructor can be overloaded.

Characteristics of Constructor (Cont..)

- It cannot be inherited (although a derived class constructor can call a base class constructor).
- Constructors cannot be virtual.
- We cannot refer to their addresses.
- Constructors make implicit call to the operators *new* and *delete* when memory allocation is required.

Note: When a constructor is declared in a class, initialization of class objects become mandatory.

Declaration of Constructor

- Declaration of constructor

```
//class with a constructor
class classname
{
  private:
    // variable and function declarations;
  public:
   // variable and function declarations;
      classname();      //constructor (having same name as the class)
};
```

Example of Constructor

```
#include<iostream>
using namespace std;
class example
{
  private:
    int a;
  public:
    example();    //constructor declared
    void display( );
};
  example:: example()  //constructor defined
     {
      a=5;
     }
  void example::display()
     {
       cout<<a;
     }
```

```
int main()
{
      example e1;                  //implicit call
      example e2=example();  //explicit call
      e1.display();
      e2.display();
      return 0;
}
```

When a class contains a constructor, it is guaranteed that an object of that class (when created) will be initialized automatically.

Not only creates the object **e1** of type **example** but also initializes its data member **a** to 5.

Types of Constructor

- Default Constructor
- Parameterized Constructor
- Copy Constructor

Default Constructor

- A constructor that accepts no parameters is called default constructor.
- The default constructor for class **example** is **example::example( )**

Parameterized Constructors

- Sometimes, it may be necessary to initialize the various data elements of different objects with different values when they are created.

- This is achieved by passing arguments to the constructor function when the objects are created.

- The constructors that can take arguments are called parameterized constructors.

Example of Parameterized Constructor

```
class example
{
  private:
    int a;
  public:
    example(int);     //Parameterized Constructor
    void display( );
};
example::example(int x)
    {
      a=x;
     }
void example::display()
     {
       cout<<a;
      }
```

When a constructor is parameterized, we must pass the initial values as arguments to the constructor function when an object is declared.

Two ways Calling:

1. Explicit

example e1 = example(5);

2. Implicit

example e1(5);

//Shorthand method

Constructors with Default Arguments

It is possible to define constructors with default arguments.

- Consider **example (int a, int b= 0);**
- The default value of the argument b is zero.

example e1(5);

assigns the value 5 to the variable a and 0 to b.

example e1(5, 3);

assigns the value 5 to the variable a and 3 to b.

Constructors with Default Arguments (Cont..)

- **example::example()** //Default Constructor
- **example::example(int a=0);** //Default Argument Constructor

 The default argument constructor can be called with either one argument or no arguments.

 When called with no arguments, it becomes a default constructor.

Copy Constructor

- A copy constructor is used to declare and initialize an object from another object.

- For example, the statement:

  example e2(e1);

  will define the object **e2** and at the same time initialize it to the value of **e1**.

- The process of initializing through a copy constructor is known as ***copy initialization***.

Example of Copy Constructor

```
#include<iostream>
using namespace std;
class example
{
  private:
    int a;
  public:
    example(int);    //Parameterized Constructor
    example(example &); //Copy Constructor
    void display();
};
example::example(int x)
    {
      a=x;
     }
example::example(example &p)
    {
      a=p.a;
     }
```

```
void example::display()
{
cout<<a;
}

int main()
{
   example e1(5);
   example e2(e1);   //or,  example e2=e1;
   e2.display();
   return 0;
}
```

OUTPUT

```
5
```

A reference variable has been used as an argument to the copy constructor.

We cannot pass the argument by value to a copy constructor.

Thank You