Constructors and Destructors

Branch: S6 ECE

Faculty: SREEDIVYA I

Constructors

- C++ provides a special member function called ‘**constructor**’ which enables an object to initialize itself when it is created.(referred to as *automatic initialization* of objects)
- The constructor has the same name as the class.
- The constructor is invoked whenever an object of its associated class is created.
- It is called constructor because it constructs the values of data members of the class
- Constructor can be defined inside or outside the class

Characteristics of a constructor

- They should be declared in the public section
- They are invoked automatically when the objects are created
- They do not have return types, not even void and therefore cannot return values
- They cannot be inherited but a derived class can call the base class constructor
- They have default arguments
- Constructors cannot be virtual
- Cannot refer to their addresses
- An object with a constructor cannot be used as a member of a union
- They make implicit calls to the new and delete operators when memory allocation is required

Example(Constructor defined inside a class)

```
#include<iostream>
using namespace std;
class integer
{
        int m,n;
        public:
          integer()
          {
           m=0;n=0;
          }
};
int main()
{
        integer int1;
        return 0;
}
```

Example(Constructor defined outside a class)

```
#include<iostream>
using namespace std;
class integer
{
        int m,n;
        public:
          integer();          // constructor declaration
};
integer::integer()                    //constructor definition
        {
        m=0;n=0;
        }
int main()
{
        integer int1;
        return 0;
}
```

- Constructors are of three types
    1. Default constructor
    2. Parameterized constructor
    3. Copy constructor

Default Constructor

- A constructor that accepts no parameters is called **default** constructor
- The default constructor for **class A is A::A()**
- If no such constructor is defined, then the compiler supplies a default constructor.
- A statement like

integer int1;

invokes the default constructor of the compiler to create the object **int1** (refer previous example for default constructor)

Parameterized Constructor

- It is possible to pass arguments to constructors.
- Typically, these arguments help initialize an object when it is created.
- To create a parameterized constructor, simply add parameters to it.
- When you define the constructor's body, use the parameters to initialize the object.
- **Uses of Parameterized constructor:**
  - It is used to initialize the various data elements of different objects with different values when they are created.
  - It is used to overload constructors.

- When the parameterized constructor is defined and no default constructor is defined explicitly, the compiler will not implicitly call the default constructor and hence creating a simple object as

  Student s;

  will return an error
- Pass the initial values as arguments to the constructor function when an object is declared
  - By calling the constructor **explicitly**
  - By calling the constructor **implicitly**

Example for parameterized constructor

```
#include<iostream>
using namespace std;
class integer
{
        int m,n;
        public:
                integer(int x, int y);
                void display(void)
                {
                        cout<<“m= ”<<m<<endl;
                        cout<<“n= ”<<n<<endl;

};
```

```
integer::integer(int x, int y)
{
m=x;
n=y;
}
int main()
{
		integer int1(10,20);		// calling constructor implicitly
		integer int2=integer(25,75);	//calling constructor explicitly
		int1.display();
		int2.display();
		return 0;
}
```

<u>OUTPUT</u>

```
m=10
n=20
m=25
n=75
```

Copy Constructor

A copy constructor is a member function that initializes an object using another object of the same class

Example

```
#include<iostream>
using namespace std;
class code
{
   int id;
   public:
     code(){}
     code(int a)
    {
      id=a;
    }
    code(code &x)
    {
      id=x.id;
    }
```

```
                    void display(void)
                    {
                          cout<<id;
                    }
};
 void main()
 {
                    code A(100);
                    code B(A);
                    code C=A;
                    cout<<"\nId of A";
                    A.display();
                    cout<<"\nId of B";
                    B.display();
                    cout<<"\nId of C";
                    C.display();
}
```

<u>OUTPUT</u>

```
Id of A 100
Id of B 100
Id of C 100
```

Constructors with default arguments

- A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the calling function doesn’t provide a value for the argument.
- In case any value is passed, the default value is overridden.
- A default constructor can either have no parameters or parameters with default arguments.

Example for constructor with default arguments

```
#include <iostream>
using namespace std;
int sum(int x, int y, int z=0, int w=0)
{
    return (x + y + z + w);
}
```

```
int main()
{
    cout << sum(10, 15) << endl;

    cout << sum(10, 15, 25) << endl;

    cout << sum(10, 15, 25, 30) << endl;
    return 0;
}
```

<u>OUTPUT</u>

```
25
50
80
```

Overloaded Constructors

- In C++, it is possible to have more than one constructor in a class with same name, as long as each has a different list of arguments. This concept is known as Constructor Overloading
- Overloaded constructors essentially have the same name of the class and different by number and type of arguments.
- A constructor is called depending upon the number and type of arguments passed.
- While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

```
#include <iostream>
using namespace std;

class construct
{
   public:
    float area;

    construct()
    {
        area = 0;
    }
    construct(int a, int b)
    {
        area = a * b;
    }

    void disp()
    {
        cout<< area<< endl;
    }
};
```

```
int main()
{
    construct o1;
    construct o2(10, 20);

    o1.disp();
    o2.disp();
    return 0;
}
```

<u>OUTPUT</u>

```
0
200
```

Destructors

- Used to destroy the objects that have been created by the constructor
- Like a constructor, destructor is a member function whose name is same as the class name but is preceded by a ‘tilde’ symbol (~)
- A destructor never takes any arguments nor it return any value.
- It will be invoked implicitly by the compiler upon exit from the program to clean up storage that is no longer needed.

```
#include <iostream>
using namespace std;
int count=0;
class alpha
{
        public:
                alpha()
                {
                   count++;
                   cout<<“\nNumber of object created:”<<count;
                }
                ~alpha()
                {
                   cout<<“\nNumber of object destroyed:”<<count;
                   count--;
                }

};
```

```
int main()
{
       cout<<“Enter Main\n”
       alpha A1,A2;
       {
        cout<<“Enter block1\n”;
        alpha A3;
       }
        return 0;
}
```

OUTPUT

```
Enter Main
Number of object created: 1
Number of object created: 2
Enter block1
Number of object created: 3
Number of object destroyed: 3
```