UNIT -2

- **Constructors in C++**

Constructor is a special method that is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally. The constructor in C++ has the same name as the class or structure. Constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object which is why it is known as constructors.

Constructor does not have a return value, hence they do not have a return type.

The prototype of Constructors is as follows:

<class-name> (list-of-parameters);

Constructors can be defined inside or outside the class declaration:-

The syntax for defining the constructor within the class:

<class-name> (list-of-parameters) { // constructor definition }

The syntax for defining the constructor outside the class:

<class-name>: :<class-name> (list-of-parameters){ // constructor definition}

Example:

// defining the constructor within the class

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

class student {
int rno;
char name[10];
double fee;

public:
student()
{
cout << "Enter the RollNo:";
cin >> rno;
cout << "Enter the Name:";
cin >> name;
cout << "Enter the Fee:";
```

```
cin >> fee;
}

void display()
{
cout << endl << rno << "\t" << name << "\t" << fee;
}
};

int main()
{
   student s; // constructor gets called automatically when
            // we create the object of the class
   s.display();

   return 0;
}
```

Output:

```
Enter the RollNo:Enter the Name:Enter the Fee:
0       6.95303e-310
```

// defining the constructor outside the class

```
#include <iostream>
using namespace std;
class student {
   int rno;
   char name[50];
   double fee;

public:
   student();
   void display();
};

student::student()
{
   cout << "Enter the RollNo:";
```

```
    cin >> rno;

    cout << "Enter the Name:";
    cin >> name;

    cout << "Enter the Fee:";
    cin >> fee;
}

void student::display()
{
    cout << endl << rno << "\t" << name << "\t" << fee;
}

int main()
{
    student s;
    s.display();

    return 0;
}
```

Output:

```
Enter the RollNo: 30
Enter the Name: ram
Enter the Fee: 20000
30 ram 20000
```

A constructor is different from normal functions in following ways:

- Constructor has same name as the class itself.
- Default Constructors don't have input argument however, Copy and Parameterized Constructors have input arguments.
- Constructors don't have return type.
- A constructor is automatically called when an object is created.
- It must be placed in public section of class.
- If we do not specify a constructor, C++ compiler generates a default constructor for object (expects no parameters and has an empty body).

Characteristics of the constructor:

- The name of the constructor is the same as its class name.
- Constructors are mostly declared in the public section of the class though it can be declared in the private section of the class.
- Constructors do not return values; hence they do not have a return type.
- A constructor gets called automatically when we create the object of the class.
- Constructors can be overloaded.
- Constructor cannot be declared virtual.
- Constructor cannot be inherited.
- Addresses of Constructor cannot be referred.
- Constructors makes implicit calls to new and delete operators during memory allocation.

- **Types of Constructors:**

1. **Default Constructors:** Default constructor is the constructor which doesn't take any argument. It has no parameters. It is also called a zero-argument constructor.

Even if we do not define any constructor explicitly, the compiler will automatically provide a default constructor implicitly.

```
// Cpp program to illustrate the
// concept of Constructors
#include <iostream>
using namespace std;

class construct {
public:
   int a, b;

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

int main()
{
```

```
    // Default constructor called automatically
    // when the object is created
    construct c;
    cout << "a: " << c.a << endl << "b: " << c.b;
    return 1;
}
```

Output:

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

2. Parameterized Constructors: 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 the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.

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 flash an error
```

```
// CPP program to illustrate
// parameterized constructors
#include <iostream>
using namespace std;

class Point {
private:
    int x, y;

public:
    // Parameterized Constructor
    Point(int x1, int y1)
    {
        x = x1;
        y = y1;
    }
```

```
    int getX() { return x; }
    int getY() { return y; }
};

int main()
{
// Constructor called
    Point p1(10, 15);

    // Access values assigned by constructor
    cout << "p1.x = " << p1.getX()
        << ", p1.y = " << p1.getY();

    return 0;
}
```

Output :

p1.x = 10, p1.y = 15

When an object is declared in a parameterized constructor, the initial values have to be passed as arguments to the constructor function. The normal way of object declaration may not work. The constructors can be called explicitly or implicitly.

Example e = Example(0, 50); **// Explicit call**

Example e(0, 50); **// Implicit call**

- **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.

3. Copy Constructor:

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

Whenever we define one or more non-default constructors (with parameters) for a class, a default constructor (without parameters) should also be explicitly defined as the

compiler will not provide a default constructor in this case. However, it is not necessary but it's considered to be the best practice to always define a default constructor.

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

Example:

```
// C++ program to demonstrate the working
// of a COPY CONSTRUCTOR
#include <iostream>
using namespace std;

class Point {
private:
    int x, y;

public:
    Point(int x1, int y1)
    {
        x = x1;
        y = y1;
    }

    // Copy constructor
    Point(const Point& p1)
    {
        x = p1.x;
        y = p1.y;
        }

    int getX() { return x; }
    int getY() { return y; }
};

int main()
{
    Point p1(10, 15); // Normal constructor is called here
    Point p2 = p1; // Copy constructor is called here

    // Let us access values assigned by constructors
    cout << "p1.x = " << p1.getX()
         << ", p1.y = " << p1.getY();
    cout << "\np2.x = " << p2.getX()
```

```
        << ", p2.y = " << p2.getY();
    return 0;
}
```

Output:

```
p1.x = 10, p1.y = 15
p2.x = 10, p2.y = 15
```

- **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.

The syntax for defining the destructor within the class

```
~ <class-name>()
{
 }
```

The syntax for defining the destructor outside the class

```
<class-name>: : ~ <class-name>(){}
```

Example:

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

class Test {
public:
    Test() { cout << "\n Constructor executed"; }

    ~Test() { cout << "\n Destructor executed"; }
};
```

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

Output:

```
Constructor executed
Destructor executed
```

Characteristics of a destructor:-

- 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.

Constructor Overloading:

In C++, We can 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** and is quite similar to function overloading.

- Overloaded constructors essentially have the same name (exact 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.

Example:

```
// C++ program to illustrate
// Constructor overloading
#include <iostream>
using namespace std;
```

```
class construct
{

public:
   float area;

   // Constructor with no parameters
   construct()
   {
      area = 0;
   }
   // Constructor with two parameters
   construct(int a, int b)
   {
      area = a * b;
   }

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

int main()
{
   // Constructor Overloading
   // with two different constructors
   // of class name
   construct o;
   construct o2( 10, 20);

   o.disp();
   o2.disp();
   return 1;
}
```

Output:

```
0
200
```