Constructors

What is the use of Constructor ?

- The main use of constructors is to initialize objects.
- The function of initialization is automatically carried out by the use of a special member function called a constructor.

General Syntax of Constructor

- Constructor is a special member function that takes the same name as the class name.
- The syntax generally is as given below:

  <class name> { arguments};
- The default constructor for a class X has the form

  X::X()

Cont……

- The constructor is automatically called when an object is created.

- There are several forms in which a constructor can take its shape namely:

✓ Default Constructor

✓ Parameterized Constructors

✓ Copy constructor

Default Constructor:

- This constructor has no arguments in it.
- Default Constructor is also called as *no argument constructor*.

Example:

```
class creature
{
    private:
    int yearofBirth;
    public:
```

Cont.....

```
creature()
        {
                cout<<“Contructor   called";
                }
         };
    int main()
    {
       creature obj;
        getch();
        return 0;
       }
```

Parameterized Constructors:

- A parameterized constructor is just one that has parameters specified in it.
- We can pass the arguments to constructor function when object are created.
- A constructor that can take arguments are called *parameterized constructors.*

Example:

```
class Creature {
private:
   int yearOfBirth;
public:
  // ...
  Creature(int year) {          //Parameterized Constructor
      yearOfBirth = year;
      }
};
```

Copy Constructor:

- Copy Constructor is used to declare and initialize an object from another object.
- For example the statement:

  abc c2(c1);

  would define the object c2 and at the same time initialize it to the value of c1.
- The process of initializing through a copy constructor is known as *copy initialization*.

Example:

```
class abc
{
      int a, b;
      public:
        abc(int x, int y)
          {
                a = x;
                b = y;
                }
                abc::abc(abc &p)
                {
                          a = p.a;
                          b = p.b;
                }
```

Cont……

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

int main()
{
    abc c1(10, 20);
    abc c2(c1);
    c1.showdata();
    c2.showdata();
    getch();
}
```

Default Arguments

- Default argument is an argument to a function that a programmer is not required to specify.

- C++ allow the programmer to specify default arguments that always have a value, even if one is not specified when calling the function.

- For example, in the following function declaration:

  int MyFunc(int a, int b, int c=12);

Cont……

- The programmer may call this function in two ways:

  result = MyFunc(1, 2, 3);

  result = MyFunc(1, 2);

- In the first case the value for the argument called c is specified as normal. In the second one, the argument is omitted, and the default value of 12 will be used instead.

- It is possible to define constructors with default arguments.

Some important points about constructors:

- Automatically called when an object is created.
- We can define our own constructors
- A constructor takes the same name as the class name.
- We can't define a constructor in the private section.

Cont.....

- No return type is specified for a constructor.
- Constructor must be defined in the public. The constructor must be a public member.
- Overloading of constructors is possible.
- If an object is copied from another object then the copy constructor is called.

Destructors

- Destructors are special member functions.
- Release dynamic allocated memory.
- Destructors are automatically named.
- Takes the same name of class name.

General Syntax of Destructors

~ classname();

Some important points about destructors:

- Take the same name as class name.
- Defined in the public.
- Destructors cannot be overloaded.
- No return type is specified.

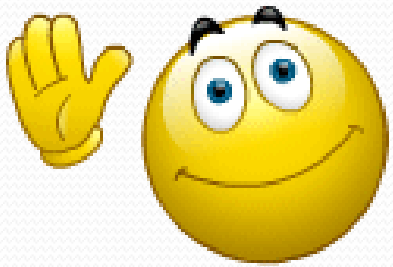
Example:

```
class creature
{
    private:
    int yearofBirth;
    public:
        creature()
        {
            yearofBirth=1970;
            cout<<"constructure called"<<endl;
            }
        ~creature()
        {
             cout<<"destructure called"<<endl;
             }
};
```

Cont……

```
int main()
  {
      cout<<"main start"<<endl;
      {
          creature obj;
      }
      cout<<"main end"<<endl;
      getch();
      return 0;
      }
```

Have a Nice Day



By Hardeep Singh