Constructors & Destructors

Constructors

A special member function having same name as that of its class which is used to initialize data members of class.

Key points while defining constructor

- A constructor has same name as that of the class to which it belongs.
- A constructor is executed automatically whenever the object is created
- A constructor doesn`t have a return type, not even void
- We can declare more than one constructor in a class. These constructor differ in there parameter list also known as constructor overloading.
- If you don’t provide a constructor of your own then the compiler generates a default constructor (expects no parameters and has an empty body).
- A constructor can preferably be used for initialization and not for input\output operations.

Contd..

- Constructor should be declared in the public section of the class. If it is not declared in public section of the class then the whole class become private . By doing this the object of the class created from outside cannot invoke the constructor which is the first member function to be executed automatically.

Syntax

```
class <class_name>
{
public:
<class_name> ([parameter list]);
};
```

Parameter list is optional.

Example

```
#include<iostream.h>
#include<conio.h>
class Rectangle
{
private:
int length,breadth;
public:
Rectangle()
{
length=5,breadth=6;
}
```

```
int area(int a,int  b)
{
int  a=(length*breadth);
cout<<“area is”<<a;
}
};
void main()
{
Rectangle r1;
r1.area();
getch();
}
```

What is the role of a constructor in classes?

a) To modify the data whenever required
b) To destroy an object
c) To initialize the data members of an object when it is created
d) To call private functions from the outer world

- TYPES OF CONSTRUCTOR:
- 1.Default Constructor
- 2.Parameterized Constructor
- 3.Copy Constructor
- 4.Dynamic Constructor

Parameterized Constructor

In order to initialize various data elements of different objects with different values when they are created. C++ permits us to achieve this objects by passing argument to the constructor function when the object are created . The constructor that can take arguments are called ***parameterized constructors***

```
class abc
{
int  m, n;
public:
abc (int x, int y); // parameterized constructor
.................
.................
};
abc : : abc (int x, int y)
{
m = x;
 n = y;
}
```

Parameterized constructor

```
#include<iostream.h>
class Rectangle
{
private:
int length,breadth;
public:
Rectangle(int a,int b)
{
length=a,breadth=b;
}
```

```
int area()
{
int  a=(length*breadth);
cout<<“area is”<<a;
}
};
void main()
{
Rectangle r1(5,6);
r1.area();
}
```

Multiple Constructors in a class

```
class rectangle {
 private:
   float height;
   float width;
   int xpos;
   int ypos;
 public:
   rectangle() { xpos = 0; ypos = 0; }
   rectangle(float, float);    // constructor
   void draw();                // draw member function
   void posn(int, int);        // position member function
   void move(int, int);        // move member function
};

rectangle::rectangle(float h, float w)
{
 height = h;
 width = w;
}
```

CONSTRUCTOR WITH DEFAULT ARGUMENTS

Just like functions, constructors can also have default values for arguments. In this case, if we are not passing the values for arguments at the time of creating objects then default value will be taken up.

Example:

```
using namespace std;
#include<iostream>
class rectangle
{
        int l,b;
        public:
                rectangle (int x=12,int y=34)
                {
                        l=x;
                        b=y;
                }
                int area()
                {
                        return(l*b);
                }
};
int main()
{
        rectangle r;
        cout<<"Area is "<<r.area();
        rectangle r1(45,67);
        cout<<"\nArea is "<<r1.area();
}
```

Copy constructor

- A copy constructor is a constructor that creates a new object using an existing object of the same class and initializes each data member of newly created object with corresponding data member of existing object passed as argument.
- since it creates a copy of an existing object so it is called copy constructor.

Example

```
class counter
{
int c;
public:
counter(int a) //single parameter constructor
{
c=a;
}
counter(counter &ob) //copy constructor
{
cout<<“copy constructor invoked”;
c=ob.c;
}
}
```

```
Void show()
{
cout<<c;
};
int main()
{
counter C1(10);
counter C2(C1);// call copy constructor
C1.show();
C2.show();
}
```

INITIALIZER LIST

- So far we have discussed constructor in which the initialization is performed in its body.
- Another alternate way is Initializer list
- Initializer list is placed between the parameter list and opening braces of the body of constructor.
- When the constructor is declared inside and defined outside the class using scope resolution then the member initialization list can only be specified within the constructor definition and not its declaration.

- An list allows initialization of data members at the time of the creation which is more efficient as values are assigned before the constructor even starts to execute.
- **rectangle(int a,int b):length(a),breadth(b){…}**

Example of initializer List

```
using namespace std;
#include<iostream>
class rectangle
{
        int l,b;
        public:
                rectangle (int x,int y):l(x),b(y){}
                int area()
                {
                        return(l*b);
                }
};
int main()
{
        rectangle r(12,34);
        cout<<"Area is "<<r.area();
}
```

What is a copy constructor?

a) A constructor that allows a user to move data from one object to another
b) A constructor to initialize an object with the values of another object
c) A constructor to check the whether to objects are equal or not
d) A constructor to kill other copies of a given object.

What happens if a user forgets to define a constructor inside a class?

a) Error occurs
b) Segmentation fault
c) Objects are not created properly
d) Compiler provides a dummy constructor to avoid faults/errors

How many parameters does a default constructor require?

a) 1

b) 2

c) 0

d) 3

Destructors

Is a member function having same name as that of constructor but it is preceded by a tilde(~) symbol and is executed automatically when object of a class is destroyed

Key points while defining destructor

- A destructor has same name as that of the class to which it belongs preceded by tilde(~)sign.
- A destructor is executed automatically whenever the object is destroyed.
- A destructor doesn`t have a return type, not even void and no arguments
- There is only one destructor in class .
- If you don’t provide a destructor of your own then the compiler generates a default destructor
- A destructor can be used to deallocate memory for an object and declared in the public section.

Need for Destructors

- To de-initialize the objects when they are destroyed
- To clear memory space occupied by a data member.

syntax

```
class CLASSNAME
{
……..……
public:
~CLASSNAME();
};
```

Example

```
#include<iostream.h>
#include<conio.h>
class counter
{
int id;
public:
counter(int i)
{
id=i;
cout<<“contructor of object with id=”<<id;
}
```

```
~counter()
{
cout<<“destructor with id=”<<id;
}
};
void main()
{
counter c1(1);
counter c2(2);
counter c3(3);
cout<<“\n end of  main”;
}
```

- Output

constructor of object with id=1

constructor of object with id=2

constructor of object with id=3

End of main

destructor with id=3

destructor with id=2

destructor with id=1