Object Oriented Programming

Constructors and Destructors

Constructors

- A special member function
- Called automatically when object of that class is instantiated/created.
- Has the same name as the class it belongs to
- No return type is used for constructors
- Constructors can be parameterized
- Constructors can be overloaded
- Purpose of Constructor
  - Perform task needed at the time of object creation like
    - Initialize data members
    - Get hold on some locks or resources(like files)
    - Allocation of run time memory

Types of constructors

- There are two types of constructors
  - Default Constructor
    - Have no parameters
  - Parameterized Constructor
    - Have parameters

Types of constructors

- Default Constructor
  - Takes no parameters
  - Example:

```
class Point{
   private:
        int x,y;
   public:
        Point(){   //Default ctor
         x = 0;
         y = 0;
         cout<<"I am default ctor of Point class"<<endl;
     }
};
```

  - An alternative way to initialize data members

```
class Point{
   private:
        int x,y;
   public:
        Point() : x(0), y(0){   //Default ctor
         cout<<"I am default ctor of Point class"<<endl;
     }
};
```

Types of constructors

- Parameterized Constructor
    - Have parameters
    - Example:

```
class Point{
   private:
        int x,y;
   public:
        Point(int a, int b){   //Parameterized ctor
         x = a;
         y = b;
         cout<<"I am parameterized ctor of Point class"<<endl;
     }
};
```

- An alternative way to initialize data members

```
class Point{
   private:
        int x,y;
   public:
        Point(int a, int b) : x(a), y(b){   //Parameterized ctor
         cout<<"I am parameterized ctor of Point class"<<endl;
     }
};
```

Constructor Overloading

- Constructors can be overloaded
  - It means we can have multiple constructors in a class
  - Objects can be instantiated in different ways

Constructor

Example program Counter

```
class Counter
{
private:
   unsigned int count;
public:
    Counter() : count(0)
     { /*empty body*/ }

    void inc_count() //increment count
    { count++; }

    int get_count() //return count
    { return count; }
};

int main()
{
    Counter c1, c2; //define and
    initialize
    cout << "\nc1=" << c1.get_count();
    cout << "\nc2=" << c2.get_count();
    c1.inc_count(); //increment c1
    c2.inc_count(); //increment c2
    c2.inc_count(); //increment c2
    cout << "\nc1=" << c1.get_count();
    cout << "\nc2=" << c2.get_count();
}
```

Output

```
c1=0
c2=0
c1=1
c2=2
```

Destructors

- A special member function
- Called automatically when object of that class is destroyed.
- Has the same name as the class it belongs to followed by a ~ sign
- No return type is used for destructors
- Destructors can not be parameterized
- Destructors can not be overloaded
- Purpose of Destructor
  - Perform task needed at the time of object killing like
    - Release resources acquired by an object
    - De-allocation of run time memory

Destructors

- An example

```
 class Counter{
    private:
       int count;
    public:
       Counter() : count(0){
     cout<<"I am ctor of Counter class"<<endl;
       }
       ~Counter(){  //Destructor of Counter class
     cout<<"I am dtor of Counter class"<<endl;
       }
 };

void main(){
   cout<<"Start of main"<<endl;
   Counter a;
   {  cout<<"Inside the block"<<endl;
      Counter b;
      cout<<"Exiting block"<<endl;
   }
   cout<<"Exiting main"<<endl;
}
```

Output

```
Start of main
I am ctor of Counter class
Inside the block
I am ctor of Counter class
Exiting block
I am dtor of Counter class
Exiting main
I am dtor of Counter class
```