Declaration and Definition of a Destructor

The syntax for declaring a destructor is :

```
-name_of_the_class()
{
}
```

So the name of the class and destructor is same but it is prefixed with a ~

(tilde). It does not take any parameter nor does it return any value. Overloading a destructor is not possible and can be explicitly invoked. In other words, a class can have only one destructor. A destructor can be defined outside the class. The following program illustrates this concept :

```
//Illustration of the working of Destructor function
#include<iostream.h>
#include<conio.h>
class add
{
     private :
      int num1,num2,num3;
     public :
     add(int=0, int=0); //default argument constructor
                     //to reduce the number of constructors void sum();
     void display();
     ~ add(void); //Destructor
};
//Destructor definition ~add()
Add:: ~add(void) //destructor called automatically at end of program
{
Num1=num2=num3=0;
Cout<<"\nAfter the final execution, me, the object has entered in the"
<<"\ndestructor to destroy myself\n";
}
//Constructor definition add()
Add::add(int n1,int n2)
{
```

```
               num1=n1;

              num2=n2;
        num3=0;
        }
        //function definition sum ()
        Void add::sum()
        {
        num3=num1+num2;
        }
        //function definition display ()
        Void add::display ()
        {
        Cout<<"\nThe sum of two numbers is "<<num3<<end1;
        }
        void main()
        {
        Add obj1,obj2(5),obj3(10,20): //objects created and initialized
 clrscr();
        Obj1.sum(); //function call
        Obj2.sum();
        Obj3.sum();
        cout<<"\nUsing obj1 \n";
        obj1.display(); //function call
        cout<<"\nUsing obj2 \n";
        obj2.display();
        cout<<"\nUsing obj3 \n";
        obj3.display();
        }
```

Some of the characteristics associated with destructors are :

(i) These are called automatically when the objects are destroyed.

(ii) Destructor functions follow the usual access rules as other member functions.

(iii) These **de-initialize** each object before the object goes out of scope.

(iv) No argument and return type (even void) permitted with destructors.

(v) These cannot be inherited.

(vi) **Static** destructors are not allowed.

(vii) Address of a destructor cannot be taken.

(viii) A destructor can call member functions of its class.

(ix) An object of a class having a destructor cannot be a member of a union.