**DESTRUCTORS**

A destructor, as the name suggests, is used to destroy the objects that have been created by a constructor. A destructor destroys the values of the object being destroyed.

A destructor is also a member functions whose name is the same as the class name but is preceded by tilde ('~').

A destructor takes no arguments and no return type can be specified for it. It is called automatically by the compiler when the object is destroyed.

**Compile time Polymorphism**

In this early binding is used in which object is bound to its function call to compile time.

For eg.

* + 1. operator overloading
    2. function overloading

**Operator Overloading**

This technique is used to assign multiple meaning to operators.

**Operators which cannot be overloaded**

* **?:** (conditional)
* **.** (member selection)
* **.\*** (member selection with pointer-to-member)
* **::** (scope resolution)
* [**sizeof**](http://en.wikibooks.org/wiki/C%2B%2B_Programming/Programming_Languages/C%2B%2B/Code/Keywords/sizeof) (object size information)

**Function overloading**

This technique is used the same function name to create function that perform a variety of different task this is known as function polymorphism.

Function overloading can be used to design a family of function with one function name by different argument list.

// Declarations

int add (int a, int b); -> 1

int add (int a, int b, int c); -> 2

double add (double X, double Y); -> 3

double add (int p, double q); -> 4

double add (double p, int q) -> 5

// function falls

cout << add (5,10); -> 1

cout << add (15,10.0); -> 4

cout << add (12.5,7.5); -> 3

cout << add (5,10,15); -> 2

cout << add (.75,5); -> 5

**Runtime Polymorphism**

Consider the situation where function name & prototype is same in both the base and drived classes.

For eg

Class A

{ int x;

public :

void show {…….} // show in base class

};

class B

{

int Y;

Public

Void show {…………….} // show in derived class

};

here the function is not overloaded and therefore static binding does not apply. We may use class resolution operator to specify the class while invoking the function with the derived class objects.

It is better if the appropriate member function could be selected while the program is running. This is known as run time polymorphism.

* This technique can be done with the help of virtual functions.

# Friend function

A friend function is used in object-oriented programming to allow access to private or protected data in a class from the outside. Normally, a function that is not a member of a class cannot access such information; neither can an external class. Occasionally, such access will be advantageous for the programmer. Under these circumstances, the function or external class can be declared as a friend of the class using the friend keyword. The function or external class will then have access to all information – public, private, or protected – within the class.

## Abstract classes

An abstract class is a class that is designed to be specifically used as a base class. An abstract class contains at least one pure virtual function. You declare a pure virtual function by using a pure specifier (= 0) in the declaration of a virtual member function in the class declaration.

The following is an example of an abstract class:

class AB {

public:

virtual void f() = 0;

};

Function AB::f is a pure virtual function. A function declaration cannot have both a pure specifier and a definition. For example, the compiler will not allow the following:

struct A {

virtual void g() { } = 0;

};

You cannot use an abstract class as a parameter type, a function return type, or the type of an explicit conversion, nor can you declare an object of an abstract class. You can, however, declare pointers and references to an abstract class.