C# delegates are similar to pointers to functions, in C or C++. A **delegate** is a reference type variable that holds the reference to a method. The reference can be changed at runtime.

Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the **System.Delegate** class.

Why do we need delegates in C#?

C# programmers often need to pass a method as a parameter of other methods when dealing with events. For this purpose, we create and use delegates in C#. A delegate is a class that encapsulates a method signature. Although it can be used in any context, it often serves as the basis for the event-handling model in C# and .NET. One good way of understanding delegates is by thinking of a delegate as something that gives a name to a method signature.

Declaring Delegates

Delegate declaration determines the methods that can be referenced by the delegate. A delegate can refer to a method, which has the same signature as that of the delegate.

For example, consider a delegate −

public delegate int MyDelegate (string s);

The preceding delegate can be used to reference any method that has a single *string* parameter and returns an *int* type variable.

Syntax for delegate declaration is −

delegate <return type> <delegate-name> <parameter list>

## Instantiating Delegates

Once a delegate type is declared, a delegate object must be created with the **new**keyword and be associated with a particular method. When creating a delegate, the argument passed to the **new** expression is written similar to a method call, but without the arguments to the method. For example −

public delegate void printString(string s);

...

printString ps1 = new printString(WriteToScreen);

printString ps2 = new printString(WriteToFile);

Following example demonstrates declaration, instantiation, and use of a delegate that can be used to reference methods that take an integer parameter and returns an integer value.

using System;

delegate int NumberChanger(int n);

namespace DelegateAppl {

class TestDelegate {

static int num = 10;

public static int AddNum(int p) {

num += p;

return num;

}

public static int MultNum(int q) {

num \*= q;

return num;

}

public static int getNum() {

return num;

}

static void Main(string[] args) {

//create delegate instances

NumberChanger nc1 = new NumberChanger(AddNum);

NumberChanger nc2 = new NumberChanger(MultNum);

//calling the methods using the delegate objects

nc1(25);

Console.WriteLine("Value of Num: {0}", getNum());

nc2(5);

Console.WriteLine("Value of Num: {0}", getNum());

Console.ReadKey();

}

}

}

## Multicasting of a Delegate

Delegate objects can be composed using the "+" operator. A composed delegate calls the two delegates it was composed from. Only delegates of the same type can be composed. The "-" operator can be used to remove a component delegate from a composed delegate.

Using this property of delegates you can create an invocation list of methods that will be called when a delegate is invoked. This is called **multicasting** of a delegate. The following program demonstrates multicasting of a delegate −

using System;

delegate int NumberChanger(int n);

namespace DelegateAppl {

class TestDelegate {

static int num = 10;

public static int AddNum(int p) {

num += p;

return num;

}

public static int MultNum(int q) {

num \*= q;

return num;

}

public static int getNum() {

return num;

}

static void Main(string[] args) {

//create delegate instances

NumberChanger nc;

NumberChanger nc1 = new NumberChanger(AddNum);

NumberChanger nc2 = new NumberChanger(MultNum);

nc = nc1;

nc += nc2;

//calling multicast

nc(5);

Console.WriteLine("Value of Num: {0}", getNum());

Console.ReadKey();

}

}

}

When the above code is compiled and executed, it produces the following result −

Value of Num: 75