Delegates in C#

A delegate is a pointer to a method. That means, a delegate holds the address of a method which can be called using that delegate.

Define a delegate

We define a delegate just like we define a normal method. That is, delegate also has a return type and parameter. For example,

```
public delegate void MyDelegate(int a, int b);
```

Any method from any class that matches the delegate signature (return type and parameter) can be assigned to the delegate.

How to store the address of a method in delegate?

Suppose we have a sum method whose signature is the same as MyDelegate as given below

```
public static void sum(int a, int b)
   {
       Console.WriteLine(a + b);
   }
```

Create an instance of MyDelegate and pass a method name as a parameter. For example,

```
MyDelegate d1 = new MyDelegate(sum);
```

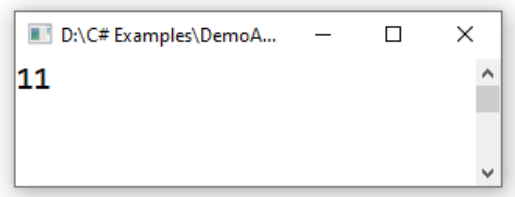
//d1 is a pointer to the method sum

Hence, we can call the method using d1 delegate reference

```
d1(5, 6);
```

Example Program

```
using System;
namespace DemoApplication
{
    class Program
    {
        public delegate void MyDelegate(int a, int b);
        public static void sum(int a, int b)
        {
            Console.WriteLine(a + b);
        }
        static void Main(string[] args) {
            MyDelegate d1 = new MyDelegate(sum);
            d1(5, 6);
            Console.ReadKey();
        }
    }
}
```



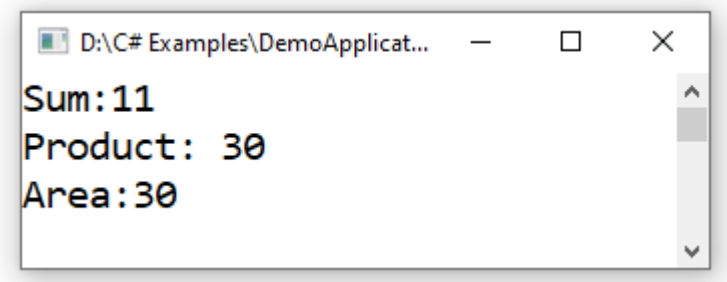
Here, we have called the sum() method by passing 5 and 6 as parameters values in d1 reference.

Multicast Delegate in C#

The multicast delegate is used to point to more than one method at a time. We use += operator to add methods to delegate. For example,

```
using System;
namespace DemoApplication
{
    class Program
    {
        public delegate void MyDelegate(int a, int b);
        public static void sum(int a, int b)
        {
            Console.WriteLine("Sum:" + (a + b));
        }
        public static void  multiply(int a, int b)
        {
            Console.WriteLine("Product: " + (a * b));
        }
        public static void RectArea(int l,int b)
        {
            Console.WriteLine("Area:" + (l*b));
        }
        static void Main(string[] args) {

            MyDelegate d1 = new MyDelegate(sum);
            d1 +=  multiply;        //multicast
            d1 +=  RectArea;
            d1(5, 6);
            Console.ReadKey();
        }
    }
}
```



Here, we have stored the multiple address of sum(), multiply() and RectArea() methods in a MyDelegate reference variable d1 using +=

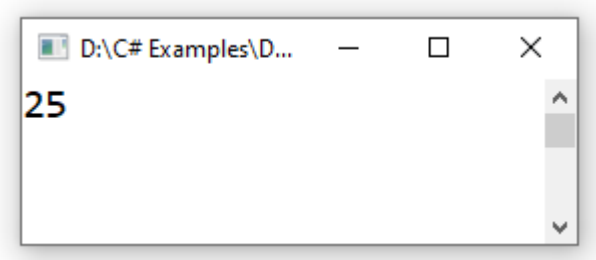
Hence, we are able to call the sum(), multiply() and RectArea() methods by passing 5 and 6 as parameters values in a single d1 reference. This is called as multicast Delegate.

Anonymous Delegates

In C#, Anonymous delegates is a pointer to refer a any method which does not have a Name and matches method signature such as the parameter and return type.

The keyword delegate is used for creating anonymous delegate.

```
using System;
namespace DemoApplication
{
    class Program
    {
        public delegate int MyDelegate(int x);
        static void Main(string[] args) {
            MyDelegate square = delegate (int x)
                {
                    return x * x;
                };
            Console.WriteLine(square(5));
            Console.ReadKey();
        }
    }
}
```



Lambda Expression in C#

C# Lambda Expression is a short block of code that accepts parameters and returns a value. It is defined as an anonymous function (function without a name). For example,

Define a Lambda Expression

We can define lambda expression in C# as,

```
(parameterList) => lambda body
```

Here,

parameterList - list of input parameters

=> - a lambda operator

lambda body - can be an expression or block of statement

The lambda expression does not execute on its own. Instead, we must use a delegate variable to represent.

Based on lambda body, the C# lambda expression is divided into two types.

1. Expression Lambda
2. Statement Lambda

1. Expression Lambda

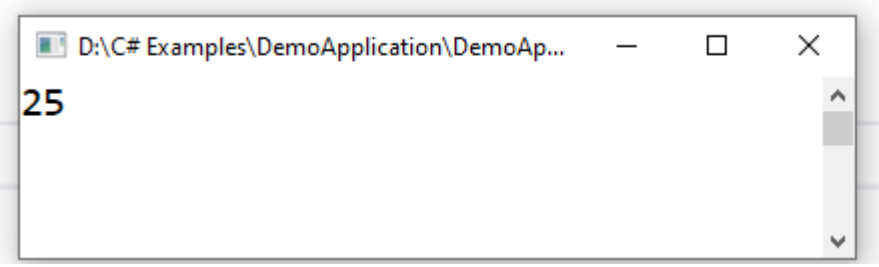
Expression lambda contains a single expression in the lambda body. For example,

```
(int num) => num * 5
```

The above expression lambda contains a single expression num * 5 in the lambda body. It takes an int input, multiplies it by 5, and returns the output.

Example Program

```
using System;
namespace DemoApplication
{
    class Program
    {
        public delegate int MyDelegate(int x);
        static void Main(string[] args) {
            MyDelegate square = (int x) => x * x;
            Console.WriteLine(square(5));
            Console.ReadKey();
        }
    }
}
```



2. Statement Lambda:

Statement lambda encloses one or more statements in the lambda body. We use curly braces {} to wrap the statements. For example,

```
(int a, int b) =>
        {
            var sum = a + b;
            return sum;
        };
```

The above expression is a statement lambda which contains two statements in the lambda body. This takes two int inputs and returns its sum.

Example Program

```
using System;
namespace DemoApplication
{
    class Program
    {
        public delegate int MyDelegate(int x);
        static void Main(string[] args) {
            MyDelegate factorial = (int num) =>
            {
                int fact = 1;
                for(int i = 2; i <= num; i++)
                {
                    fact = fact * i;
                }
                return fact;
            };
            Console.WriteLine("Factorial: " + factorial(5));
            Console.ReadKey();
        }
    }
}
```