MCSE 541: Web Computing and Mining

C# Delegates, Anonymous Function & Lambda Expressions

Prof. Dr. Shamim Akhter
shamimakhter@iubat.edu

@Text Book: Chapter 9,10

Function Pointer

```
#include<stdio.h>

void fun(int a){
        printf("Value of a is%d\n",a);
}

int main(){
        void (*fun_ptr)(int);
        fun_ptr=&fun;

      (*fun_ptr) (10);
       return;
}
```

- Output: Value of a is 10

- Void is considered a data type (for organizational purposes), but it is basically a keyword to use as a placeholder where you would put a data type, to represent "no data".
- void is an incomplete type that cannot be completed, This means you cannot apply the sizeof operator to void, but you can have a pointer to an incomplete type.
- Hence, you can declare a routine which does not return a value as: **void MyRoutine();**
- But, you cannot declare a variable like this: **void bad_variable;**
- However, when used as a pointer, then it has a different meaning: void* vague_pointer;
- This declares a pointer, but without specifying which data type it is pointing to.

Steps:

1. We define a function pointer(fp).
2. Assign the value to fp.
3. Call function pointer(fp).

Delegates

- Delegates is an object that refer to a method.
- Same delegates can refer to call different methods.
  - Which method a delegate will refer decided in runtime

General form for delegate

delegate *ret-type* name(*parameter-list*);

- Delegate ret-type is the reference method ret-type
- Parameter-list depends on the method argument list
- Delegate can not be static but delegate ref can be static

- Why delegates:
  - Delegates support events.
  - Delegates give your program a way to execute methods at runtime without having to know precisely what those methods are at compile time.

Delegates Example

```
using System;
public delegate int Mydelegate(int x,int y);

class sample
{
    public static int rectangle(int a, int b)
    {
            return a * b;
    }
}
```

There are three(3) steps involved while working with delegates:
- Declare a delegate,
- Set a target method, and
- Invoke a delegate.

```
class Program
{
    static void Main()
    {
      Console.WriteLine("My simple Delegate Program");
      Mydelegate mdl = new Mydelegate(sample.rectangle);
      Console.WriteLine("The Area of rectangle is {0}", mdl(4, 5));
      Console.ReadKey();
    }
}
```

Example

```
using System;
public delegate string StrMod(string str);
class DelegateTest{
        public static string ReplaceSpaces(string s){
                Console.WriteLine("Replacing spaces with hyphens.");
                return s.Replace(' ', '-');
        }
        public static string RemoveSpaces(string s){
                string temp=" ";    int i;
                Console.WriteLine("Removing spaces.");
                for(i=0; i<s.Length; i++)
                        if(s[i]!=' ') temp+=s[i];
                return temp;
        }
        public static string Reverse(string s){
                string temp="";        int j;
                Console.WriteLine("Reversing string.");
                for(j=s.Length-1; j>=0; j--)
                        temp+=s[j];
                return temp;
        }
}
```

```
class Test{
   public static void Main(){

            string str1, str2, str3;
            StrMod strOp = new StrMod(DelegateTest.ReplaceSpaces);
            str1=strOp("This is the Test.");
            Console.WriteLine("Resulting string: "+str1);

            strOp = new StrMod(DelegateTest.RemoveSpaces);
            str2=strOp("This is the Test.");
            Console.WriteLine("Resulting string: "+str2);

            strOp = new StrMod(DelegateTest.Reverse);
            str3=strOp("This is the Test.");
            Console.WriteLine("Resulting string: "+str3);

   }
}
```

Method Group Conversation 2.0 C#

```
class Test{
    public static void Main(){
        string str1, str2, str3;
        StrMod strOp = DelegateTest.ReplaceSpaces;
        str1=strOp("This is the Test.");
        Console.WriteLine("Resulting string: "+str1);

        strOp = DelegateTest.RemoveSpaces;
        str2=strOp("This is the Test.");
        Console.WriteLine("Resulting string: "+str2);

        strOp = DelegateTest.Reverse;
        str3=strOp("This is the Test.");
        Console.WriteLine("Resulting string: "+str3);
    }
}
```

Multicasting-invocation list/chain

```
class Test{
    public static void Main(){
            string str, str1;
            str1="This is the Test.“;
            StrMod strOp;
            StrMod strOp1 = DelegateTest.ReplaceSpaces;
            StrMod strOp2 = DelegateTest.RemoveSpaces;
            StrMod strOp3 = DelegateTest.Reverse;
            strOp = strOp1;
            strOp += strOp2;
            strOp +=strOp3;
            str=strop(str1);
            Console.WriteLine("Resulting string: "+str);
    }
}
```

Multicasting-invocation list/chain With ref variable

```
delegate void StrMod(ref string s);

class sample {
        public static void ReplaceSpaces(ref string s){
            string temp;
            Console.WriteLine("Replacing spaces with hyphens.");
            temp= s.Replace(' ', '-');
            s = temp;
        }
        public static void RemoveSpaces(ref string s){
            string temp = " "; int i;
            Console.WriteLine("Removing spaces.");
            for (i = 0; i < s.Length; i++)
                if (s[i] != ' ') temp += s[i];
            s=temp;
        }
    }
```

```
public static void Reverse(ref string s)
{
        string temp = ""; int j;
        Console.WriteLine("Reversing string.");
        for (j = s.Length - 1; j >= 0; j--)
            temp += s[j];
        s= temp;
}
```

```
class Program
    {
        static void Main(string[] args)
        {
            String str = "This is a pot.";
            StrMod sd1, sd2, sd3;

            sd1= sample.ReplaceSpaces;
            sd2 = sample.RemoveSpaces;
            sd3 = sample.Reverse;

            sd1 += sd2;
            sd1 += sd3;

            sd1(ref str);
            Console.WriteLine(str);
        }
    }
```

Anonymous Function

Nameless methods.

They prevent creation of separate methods, especially when the functionality can be done without a new method creation.

Anonymous methods provide a cleaner and convenient approach while coding.

- A method can be used only by its delegate
  - Invoke via a delegate and never call on its own
- That time anonymous function is useful
  - Unnamed block of code that is passed to a delegate constructor
- Two types (C# 3.0) of Anonymous Function:
  - anonymous method and lambda expression

Delegates and Anonymous Method

```
class Program
 {
      delegate void Countit();
       static void Hello()
       {   Console.WriteLine("Hello World!"); }

       static void Main(string[] args)
       {
           //Countit count = new Countit(Program.Hello);
           //count();
           Countit count = delegate { Console.WriteLine("Hello World!");
                                      }; //Anonymous Method
           count();
      }
  }
```

Anonymous method with argument and return values

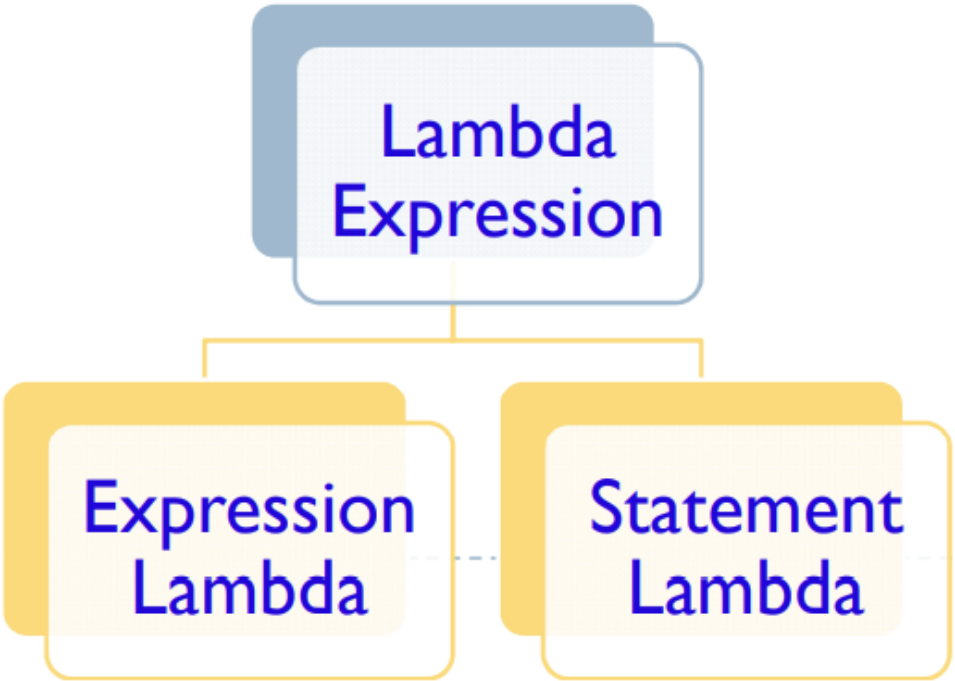
```
class Program
    {
        delegate int Countit(int end);
        static void Main(string[] args)
        {
            Countit count = delegate(int end) {
                                int i, sum = 0;
                                for (i = 1; i <= end; i++)
                                    sum += i;
                                return sum;
                            }; //Anonymous Method

            Console.WriteLine(count(5));
        }

    }
```

Lambda Expression

- An alternative to anonymous method.
- Mostly uses in LINQ (Language Integrated Query)
  - applicable to delegates and events
- Lambda Expressions use lambda operator => (goes, becomes)
  - it divides the expression into two parts
  - left is specified with input parameter, right is the lambda body



Expression Lambda

Parameter(s)	=>	Expression

- Expression on the right side of Lambda, acts on parameter(s) on the left side of Lambda.

- count=> count+2; return the value of count increased by two
- n => n%2 ==0; return true if n is even false if n is odd

3. Declare a delegate and show how a delegate replaces the following method.

```
public static double FunctionName (int b){
        return (1.0* b+2);
}
```

```
public delegate double MyD (int b)
MyD D = new MyD (FunctionName);
D(100);
```

4. Convert the following Anonymous function represented with Lambda expression to Anonymous function representation with delegate representation.

```
Delegate-D D = (int n) => {
                int c=500;
                Console.WriteLine(c);
                return(n+c);
                };
```

```
Delegate-D D = delegate (int n)
                { int c=500;
                Console.WriteLine(c);
                return(n+c);
                };
```

5. Write the code for defi...

4. Coverts the following anonymous class to Lambda Expression.

```
p.SomeEvent += delegate (int n){
        Console.WriteLine("Event Occured for "+n);
};
```

```
p.SomeEvent +=
(int n) => {
        Console.WriteLine("
        Event Occured for"+
        };
```