Introduction

In programming, a function is defined as named code block that can perform a specific task. Some programming languages make a distinction between a function, which returns a value, and a procedure, which performs some operation but does not return a value. C# does not seem to make such distinction. In C#, a function is a way for programmers to organize codes into a code block that does something or returns a value.

Previous lectures showcased sample codes that use a variety of methods provided by the .Net Framework, such as the “Sqrt()” method of the Math class as shown below. The term “method”, which is typically used in object-oriented programming, refers to a function that is internal to part of a class. In other words, a “method” is a function of a class.

```csharp
using System;
using System.Windows.Forms;

class Example
{
    public static void Main()
    {
        MessageBox.Show(Math.Sqrt(17.65) + "");
    }
}
```

The above example also shows that all C# programs must have a method named “Main()”, which is the starting point for the program. “Main()” is a method because it is a function of the “Example” class. When the application is started, the “Main()” method is the first method that is invoked.

Since there can only be one entry point in a C# program, programmers can add other methods, known as user-defined functions. Technically speaking, all user-defined functions in C# are member function of a class. Yet, interestingly, they are commonly called functions.

User-defined functions

In programming, a user-defined function is a named section of code that performs a specific task. The following illustrates the syntax to create a user-defined function in which “static” is a keyword that requires students’ attentions. The “static” keyword is used to mark the function as “non-instantiable” which means the function can be accessed without creating an instance of the class. The code block function is enclosed by a pair of curly brackets, { and }.

```
accessModifier [static] returnType functionName(parameters) { }
```

Access modifiers are keywords used to specify the accessibility of a function. There are three commonly used access modifiers: public, protected, and private. A function without designated modifier is assumed public by default.

```
private string getDateDateTime() { }
```

or simply,

```
string getDateDateTime() { }
```

The following table distinguishes these access modifiers.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>anyone can access</td>
</tr>
<tr>
<td>protected</td>
<td>only within the class and derived classes</td>
</tr>
<tr>
<td>private</td>
<td>only within the class</td>
</tr>
</tbody>
</table>

The following table distinguishes these access modifiers.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>anyone can access</td>
</tr>
<tr>
<td>protected</td>
<td>only within the class and derived classes</td>
</tr>
<tr>
<td>private</td>
<td>only within the class</td>
</tr>
</tbody>
</table>
A private function is often used as a supportive function that provides a value other program entity in the class. In the following example, “getValue()” is a supportive function that performs the calculation and returns the results to its calling party. By the way, MessageBox.Show() is the calling party.

```csharp
using System;
using System.Windows.Forms;

class Example
{
    private static double getValue()
    {
        return Math.Sqrt(17.65);
    }

    public static void Main()
    {
        MessageBox.Show(getValue() + "");
    }
}
```

They **return type** specifies the type of data the function can return to its calling party. The term “calling part” refers to the code segment that instructs (or calls) the function to perform the task. Any primitive type, such as `int`, `double`, `string`, `char`, and `bool`, as well as `void`, can be the return type. Interestingly, the “void” type is the only one that instructs the function not to return any value to its calling party. A later section will discuss this concept in detail. The following demonstrates how to declare a “void” function.

```csharp
public void getDateDateTime() { }
```

The following demonstrates how to declare a function that will return a value of `int` type. By the way, the code block requires the use of “return” keyword to return a value (or an expression that can generate a value).

```csharp
public int getDateDateTime()
{
    ........
    return value;
}
```

The following is a sample value-returning function, named “getValue()”. It returns a value of `double` type.

```csharp
private static double getValue()
{
    return Math.Sqrt(17.65);
}
```

The term **“parameter”** describes variables of the function. They are enclosed in parentheses and are separated by commas. In the following example, “d”, “m”, and “y” are three variables of `int` type. They are variables of the function and are used to keep the values passed to the function.
public void getDateTime(int d, int m, int y) {}  

Empty parentheses indicate that the method requires no parameters.

public void getDateTime(DateTime dt) {}  

The following contains a getDateTime() function that returns the current date and time values as a string (through a conversion done by the Convert.ToString() method) to its calling party. When a segment of code calls a function to perform a special task, it is known as the “calling party”. In the following code, the Show() method is the calling party.

using System;
using System.Windows.Forms;
class MyFunc {
static void Main()
{
   MessageBox.Show(getDateTime());
}
private static string getDateTime()
{
   return Convert.ToString(DateTime.Now);
}
}

It is necessary to note that functions that operate on instances of the class and is known as an instance function. C# function has a fundamentally different design paradigm than C and its descendants. C# function strictly follows object-oriented paradigm. C# instance functions are not independent entities. They are, by default, a member of the class. There is a need to create an instance of the class and then use it as a reference to call the function. In the following example, the “getDateTime()” function is declared without the “static” keyword. Therefore, an instance of the “MyFunc” class must be created in order to call the “getDateTime()” function using objectID.functionname format. By the way, “m1” is the identifier of the object.

using System;
using System.Windows.Forms;
class MyFunc {
static void Main()
{
   MyFunc m1 = new MyFunc();
   MessageBox.Show(m1.getDateTime());
}
private string getDateTime()
{
   return Convert.ToString(DateTime.Now);
}
}

In C and its descendant languages, functions are independent entities and can be called from anywhere in the program. However, C# requires the use of the static keyword if developers want the function to be independent from the class. The following is another example that demonstrates how the getDateTime() function is declare with a static keyword to bypass the object-oriented paradigm.
using System;
using System.Windows.Forms;

class MyFunc
{
    static void Main()
    {
        MessageBox.Show(getDateTime());
    }

    static string getDateTime()
    {
        return Convert.ToString(DateTime.Now.ToShortDateString());
    }
}

By the way, the DateTime class provides many tools to implements the date and time. The following are few methods for defining the format of output. See http://msdn.microsoft.com/en-us/library/system.datetime.aspx for a list of available tools.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToLongDateString()</td>
<td>Converts the value of the current DateTime object to long date string representation.</td>
</tr>
<tr>
<td>ToLongTimeString()</td>
<td>Converts the value of the current DateTime object to long time string representation.</td>
</tr>
<tr>
<td>ToShortDateString()</td>
<td>Converts the value of the current DateTime object to short date string representation.</td>
</tr>
<tr>
<td>ToShortTimeString()</td>
<td>Converts the value of the current DateTime object to short time string representation.</td>
</tr>
</tbody>
</table>

There is a shorthand way to create an instance of the class and immediately use it to reference non-static functions. The following is the syntax, in which new is a keyword.

(new className()).functionName()

A complete sample code is:

    using System;
    using System.Windows.Forms;

    class MyFunc
    {
        static void Main()
        {
            MessageBox.Show((new MyFunc()).getDateTime());
        }

        private string getDateTime()
        {
            return Convert.ToString(DateTime.Now);
        }
    }

Concept of returning type

A function that returns a value to its calling party is typically considered as a supportive function that performs a particular task for the calling party. They are frequently declared as private; thus,
they can be accessed only by code in the same class. A class can contain many supportive functions.

As stated previously, C# functions can be categorized as value-returning and void types. The main difference is that a value-returning function must return a value to its calling party and the returned value must be the same data type as the function, while a void function does not return anything to its calling party. The following compares a value-returning function to a void function. They produced the same results although their programming logic differs.

| value-returning | using System;  
|                 | using System.Windows.Forms;  
|                 | class MyFunc  
|                 | {  
|                 | static double getSqrt()  
|                 | {  
|                 | return Math.Sqrt(44.98);  
|                 | }  
|                 | static void Main()  
|                 | {  
|                 | MessageBox.Show(getSqrt() + "");  
|                 | }  
| void             | using System;  
|                 | using System.Windows.Forms;  
|                 | class MyFunc  
|                 | {  
|                 | static void getSqrt()  
|                 | {  
|                 | MessageBox.Show(Math.Sqrt(44.98) + ");  
|                 | }  
|                 | static void Main()  
|                 | {  
|                 | getSqrt()  
|                 | }  

In the case of value-returning, the “getSqrt()” function is used as supportive function that calculates the square root of 44.98 for the MessageBox.Show() method. In the case of void, is “getSqrt()” function is called to perform the task -- calculate the square root and display the result without sending anything back to its calling party.

The following figure illustrates the design concept of an application. The “Main()” method will call each of the three supportive functions. All of the three supportive functions are value-returning type.

In the following example, the instructor declares a DateTime instance named “dt” and uses it to reference members of the DateTime structure of .NET Framework. The Year() function returns an int type of value which is the current year value provided by the Year property of DateTime.
structure. The `leap()` function returns a Boolean value: `true` means a leap year, `false` mean not a leap year. The checking is done by the `IsLeapYear()` method of `DateTime` structure. The `Ampm()` function returns a `string` value. This function will retrieve the current hour value and then check if it is greater than 12, return “am” if so; otherwise, return “pm”.

```csharp
using System;
using System.Windows.Forms;

class MyFunc
{
    static DateTime dt = DateTime.Now;
    static void Main()
    {
        string str = "Current year: " + Year();
        str += "\nLeap year: " + leap();
        str += "\nAM/PM: " + Ampm();
        MessageBox.Show(str);
    }
    private static int Year()
    {
        return dt.Year; // get current year value
    }
    private static bool leap()
    {
        return DateTime.IsLeapYear(dt.Year);
    }
    private static string Ampm()
    {
        return (dt.Hour > 12 ? "am" : "pm"); // PM starts at 12:00
    }
}
```

A sample output looks:

![Screenshot of output](image)

The above code also uses the **conditional operator** (`?:`). The conditional operator (`?:`) returns one of two values depending on the value of a Boolean expression. The syntax is:

```
condition ? outputIfTrue : outputIfFalse;
```

Once again, the following statement will check if the value of “dt.Hour” is greater than 12, return “am” if so; otherwise, return “pm”.

```
dt.Hour > 12 ? "am" : "pm"
```

A **void** function means it does not have to return anything to its calling party. Functions of void type usually are not supportive functions. They are designed for a theme or a special feature of the application. In the following code, the “Main()” method will call either the “showIt()” or the
“msg()” function for execution depending on the condition of \((x \neq 0)\); therefore, the “Main()” is the calling party.

```csharp
using System;
using System.Windows.Forms;

class MyFunc
{
    static int x = (new Random()).Next(0, 6);

    static void Main()
    {
        if (x != 0)
        {
            showIt();
        }
        else
        {
            msg();
        }
    }

    private static void showIt()
    {
        MessageBox.Show("x is "+x+"nx / 3 = "+(float)x/3);
    }

    public static void msg()
    {
        MessageBox.Show("Error: Nothing can be divided by zero.");
    }
}
```

The “Main()” method will call the “showIt()” methods if \(x\) is not equal 0; otherwise, it will call the “msg()” method to display an error message. Neither of these two functions will return a value to the “Main()” method. Yet, they provide features to the application.

Function of void type can have their supportive functions, too. The following figure illustrates the design logic. The Main() method call the getHour(). The getHour() function then calls its supportive function, getUTCHour().

```csharp
using System;
using System.Windows.Forms;

class MyFunc
{
    static int utcH = 0;

    static void Main()
    {
        getHour();
    }

    public static void getHour()
    {
        getUTCHour();
    }
}
```

The following is the complete code. The “getHour()” function is the calling party of “getUTCHour()” which changes the value the “utcH” variable.

```csharp
using System;
using System.Windows.Forms;

class MyFunc
{
    static int utcH = 0;

    static void Main()
    {
        getHour();
    }

    public static void getHour()
    {
        getUTCHour();
    }
```
In addition to the primitive types, array of a primitive type can be the return type of a C# function. In the example, the “squareIt()” function has the return type int[].

```csharp
public static int[] squareIt() { }
```

In the following example, the “squareIt()” function converts every element of the array (“a”) to their squares, and then return the processed int array to its calling party which is another array “b”.

```csharp
using System;
using System.Windows.Forms;
public class Example
{
    static int[] a = new int[7] {8, 5, 3, 2, 4, 9, 6}; // original
    public static int[] squareIt()
    { 
        for (int i=0; i<7; i++)
        { 
            a[i] = a[i] * a[i]; 
        }
        return a;
    }
    static void Main()
    { 
        int[] b = squareIt(); // squared
    }
}
```

The following is a sample code that works on an int array and returns an int value (which is the sum of all elements in the int array) as output.

```csharp
using System;
using System.Windows.Forms;
public class Example
{
    static int[] x = new int[7] {8, 5, 3, 2, 4, 9, 6}; // original
    public static int Sum()
    { 
        int s = 0;
        for (int i=0; i<x.Length; i++)
        { 
            s += x[i]; 
        }
        return s;
    }
}```
return s;
}

static void Main()
{
    MessageBox.Show(Sum() + "");
}

The following is another example. It calls the “capIt()” function to break a string “apple” into an array of char and use a for loop to convert every element from lowercase to uppercase. Finally, it returns the converted char array. By the way, “capIt()” is declared to return char[] type.

using System;
using System.Windows.Forms;

public class Example
{
    static string s = "apple";
    static char[] c = new char[5];

    public static char[] capIt()
    {
        for (int i=0; i<s.Length; i++)
        {
            c[i] = Convert.ToChar(s[i].ToString().ToUpper());
        }
        return c;
    }

    static void Main()
    {
        c = capIt();
    }
}

Parameters

A function can have its variable(s) (singular or plural), known as parameters, for its calling party to pass value(s) to be processed by the function. The following illustrates the syntax to define parameters in C#.

    functionname(datatype param1, datatype param2, ...)

In the following example, the “getSqrt()” function takes one parameter named “d” which is the variable of the “getSqrt()” function for keeping the value passed by the calling party. By the way, the calling party passes a value 58.69.

using System;
using System.Windows.Forms;

class MyFunc
{
    static double getSqrt(double d)
    {
        return Math.Sqrt(d);
    }

    static void Main()
    {
        MessageBox.Show(getSqrt(58.69) + "");
    }
}
The following declares the “getSqrt()” function as void type. It also demonstrates how to pass two values in different types as parameters.

```csharp
using System;
using System.Windows.Forms;

class MyFunc
{
    static void getSqrt(double d, int theta)
    {
        MessageBox.Show((Math.Sqrt(d) - Math.Cos(theta)) + "");
    }

    static void Main()
    {
        getSqrt(58.69, 75);
    }
}
```

In the following example, the “findIt()” function requires a variable of DateTime type as its parameter; therefore, its calling party must pass a value of DateTime type to it. By the way, the parameter is assigned an identifier “dt”.

```csharp
using System;
using System.Windows.Forms;

class MyFunc
{
    static void Main()
    {
        findIt(new DateTime(2018, 12, 25)); // Dec, 25, 2018
    }

    public static void findIt(DateTime dt)
    {
        MessageBox.Show("It is a " + dt.DayOfWeek);
    }
}
```

The output looks:

![Message Box]

The above code declares the “findIt()” method as `void` type. The following is the value-returning type version of the code which produces the same result as the above.

```csharp
using System;
using System.Windows.Forms;

class MyFunc
{
    static void Main()
    {
```
```csharp
    MessageBox.Show("It is a " + findIt(new DateTime(2018, 12, 25)));

    public static string findIt(DateTime dt)
    {
        return Convert.ToString(dt.DayOfWeek);
    }
```

The following example is another version of the above code. The `findIt()` function takes three `int` parameters: y, m, and d. This code produces the same result as the above.
```csharp
    using System;
    using System.Windows.Forms;

    class MyFunc
    {
        static void Main()
        {
            findIt(2018, 12, 25);  // Dec, 25, 2018
        }

        public static void findIt(int y, int m, int d)
        {
            DateTime dt = new DateTime(y, m, d);
            MessageBox.Show("It is a " + dt.DayOfWeek);
        }
    }
```

The value-returning version of the above code is shown below for the sake of comparison.
```csharp
    using System;
    using System.Windows.Forms;

    class MyFunc
    {
        static void Main()
        {
            MessageBox.Show("It is a " + findIt(2018, 12, 25));
        }

        public static string findIt(int y, int m, int d)
        {
            DateTime dt = new DateTime(y, m, d);
            return Convert.ToString(dt.DayOfWeek);
        }
    }
```

A parameter of a function can be an array. In the following code, the `findMin()` function takes an `int` array passed by the `str` variable of the `Main()` method. This function then uses a simple algorithm to find the minimal elements and return it to the calling party. The algorithm is: start with assuming `x[0]` is the minimum and assign it to a variable named “`min`”, compare `x[i]` with “`min`”, assign the value of `x[i]` to “`min`” only when `x[i]` is found smaller than “`min`”, complete the iteration.
```csharp
    using System;
    using System.Windows.Forms;

    class myMin
    {
        private static int findMin(int[] x)
```
{ int min = x[0];

for (int i=1; i<x.Length; i++)
{
    if (x[i]<min) { min = x[i];} 
}

return min;
}

static void Main()
{
    int[] x = new int[15];
    Random rn = new Random();
    string str = "The list is:\n";
    for (int i=0; i<x.Length; i++)
    {
        x[i] = rn.Next(10, 100);
        str += x[i] + " ";
    }

    // call the function here
    str += "\nThe min is : " + findMin(x);
    MessageBox.Show(str);
}
}

In the following code, there are two methods -- Square() and Cube() -- in addition to the Main() methods. They both have a parameter x of double type. They are declared with a return type, double, and are the so-called value-returning type of functions. Therefore, they must return a double value to their calling party (in this code, it is the str variable). The "return" keyword is the one that actually returns the calculation results to the calling party.

using System;
using System.Windows.Forms;

class Lab2_0
{
    static void Main(string[] args)
    {
        String str = "The square of 3.1 is " + Square(3.1) + "\n";
        str += "The cube is " + Cube(3.1);
        MessageBox.Show(str);
    }

    static double Square(double x)
    {
        return x * x;
    }

    static double Cube(double x)
    {
        return x * x * x;
    }
}

The output looks:
In the Main() method, the following bold-faced code segment “calls” the Square() method and passes a value 3.1 to the x parameter of the Square() method.

```csharp
String str = "The square of 3.1 is " + Square(3.1) + "\n";
```

The Square() method performs the calculation 3.1×3.1 due to the following definition and then returns the calculation results, 9.61, to the str variable of Main() method. “*” is the multiplication operator. A later lecture will discuss about it.

```csharp
static double Square(double x)  
{  
    return x * x;  
}
```

The following is a sample void type version of the above code.

```csharp
using System;  
using System.Windows.Forms;  

class MyFunc  
{  
    static string str = "";  
    static void Main(string[] args)  
    {  
        str += "The square of 3.1 is ";  
        Square(3.1);  
        str += "\nThe cube is ";  
        Cube(3.1);  
        MessageBox.Show(str);  
    }  
    static void Square(double x)  
    {  
        str += x * x;  
    }  
    static void Cube(double x)  
    {  
        str += x * x * x;  
    }  
}
```

**Reusability**

A function can perform the task routinely on demand. Functions are said to be reusable because they can be used repeatedly in a program. In the following code, the str variable of the Main() method calls the “getIt()” function five times. The “getIt()” function returns a random number to its calling party (str) each time when a calling occurs.

```csharp
using System;  
using System.Windows.Forms;
```
class MyFunc
{
    Random rn = new Random();

    static void Main()
    {
        MyFunc m = new MyFunc();
        string str = m.getIt() + "\n";
        str += m.getIt() + "\n";
        str += m.getIt() + "\n";
        str += m.getIt() + "\n";
        str += m.getIt() + "\n";
        MessageBox.Show(str);
    }

    public int getIt()
    {
        return rn.Next(100);
    }
}

The following is another example to demonstrate the reusability of functions in C#. It uses a for loop to call the functions several times.

using System;
using System.Windows.Forms;

class MyFuc
{
    static void Main()
    {
        string str = "";
        for (int i=0; i<5; i++)
        {
            str += getDateTime() + "\n";
        }
        MessageBox.Show(str);
    }

    static string getDateTime()
    {
        return Convert.ToString(DateTime.Now);
    }
}

Functions can be a reusable subroutine because it can individually perform a special task. In the following example, the findMax() function takes a double array and uses an algorithm to find the maximal value from the array. The algorithm is simple: start with assuming d[0] is the maximum by assigning d[0] as value of a variable max, use a for loop to compare max with every d[i], assign the value of d[i] to max only when d[i] is found larger than max, and then return the maximum to calling party. The createIt() function create another double array and then uses a for loop to randomly generate 100 values as elements of the double array for its calling party. There is a default constructor that creates an input box to collect inputs. A default constructor is the constructor in a class that has exactly the same identifier with the class. When
an instance of the class is invoked, the default constructor executes automatically. By the way, a later lecture will discuss basic object-oriented programming using C# in detail.

```csharp
using System;
using System.Windows.Forms;

public class Example
{
    static string lst = ""
    static Random rn = new Random();
    static double[]} x;

    private static double findMax(double[] d)
    {
        double max = d[0];
        for (int i=1; i<d.Length; i++)
        {
            if (d[i]>max) { max = d[i]; }
        }
        return max;
    }

    private static double[] createIt()
    {
        double[] d = new double[100];
        lst = "The list is: \n"
        for (int i=0; i<d.Length; i++) // generate 47 numbers
        {
            // Math.round(value, 2) round to 2nd digit after decimal point
            d[i] = Math.Round(rn.NextDouble() * 10, 2);
            lst += d[i] + " ";
        }
        return d;
    }

    private MyClass() // default constructor
    {
        int y = Convert.ToInt32(InputBox.Show("How many lists?"));
        string str = "Number of list(s): " + y;
        for (int i=0; i<y; i++)
        {
            x = createIt(); // create the array
            str += "\n\n" + lst + "\nMax is " + findMax(x); // get max
        }
        MessageBox.Show(str);
    }

    static void Main()
    {
        new MyClass();
    }
}
```

There are two functions and one default constructor in the above code. Each of them is responsible for a special task, although the default constructor will call the other two functions.
The Main() method simply creates the instance. In the above code, there is only one instance. However, it will create three instances if the code changes to:

```csharp
static void Main()
{
    new myClass();
    new myClass();
    new myClass();
}
```

To compile the above code, use: `csc /t:winexe filename.cs InputBox.cs`.

A C# **delegate** is a type that defines a method or function signature. According to Microsoft, “the signature of a method consists of the name of the method and the type and kind (value, reference, or output) of each of its formal parameters, considered in the order left to right. The signature of a method specifically does not include the return type, nor does it include the `params` modifier that may be specified for the right-most parameter.”

Delegates are used to pass methods as arguments to other methods. Using a delegate allows the programmer to encapsulate a reference to a method inside a delegate object. The delegate object can then be passed to code which can call the referenced method, without having to know at compile time which method will be invoked. According to Microsoft, delegates are object-oriented, type-safe, and secure. The following example shows a delegate declaration. The name of delegate is “Dg”. It can encapsulate a function that takes two `int` data as parameters and return another `int` data.

```csharp
public delegate int Dg(int x, int y);
```

Students can create a function as the “**delegate object**” which means the delegate will represent the function. The following creates a function named “**Sum**” to be the “delegate object”.

```csharp
public static int Sum(int x, int y)
{
    return x + y;
}
```

Once both the delegate and delegate object are defined, programmers can use the following statement to build the delegation relationship. Technically speaking, the following statement declares an instance of “Dg” with “handle” as its identifier. It also assigns “Sum” as the delegate object.

```csharp
Dg handle = Sum;
```

Once a delegate is instantiated, a call made to the delegate will be passed by the delegate to that function. In the following statement, “handle” is the delegate of “Sum”; therefore, handle(2, 5) is equivalent to Sum(2, 5).

```csharp
MessageBox.Show(handle(2, 5) + "");
```

The complete code is:

```csharp
using System;
using System.Windows.Forms;

class MyDel
{
    public delegate int Dg(int x, int y); // declare a delegate
    
    public static int Sum(int x, int y) // delegate object
    {
    }
}
```
```csharp
{ return x + y;
}

static void Main(string[] args)
{
    Dg handle = Sum; // instantiate the delegate
    MessageBox.Show(handle(2, 5) + "");
}

Another version of the above example is:
using System;
using System.Windows.Forms;

class MyDel
{
    public delegate void Dg(int x, int y); // declare a delegate
    public static void Sum(int x, int y) // delegate object
    {
        MessageBox.Show((x + y) + ");
    }
    static void Main(string[] args)
    {
        Dg handle = Sum; // instantiate the delegate
        handle(2, 5);
    }
}

After instantiating a delegate, developers can associate its instance with any function as long as their structure and type are compatible. In other words, the identifier of the delegate (such as “handle”) is transferrable. In the following, the identifier “handle” is the delegate of “Sum”, yet it later delegates “Product”; therefore, both “Sum” and “Product” are delegate objects of “Dg”.
using System;
using System.Windows.Forms;

class MyDel
{
    public delegate int Dg(int x, int y); // declare a delegate
    public static int Sum(int x, int y) // delegate object
    {
        return (x + y);
    }
    public static int Product(int x, int y)
    {
        return (x * y);
    }
    static void Main(string[] args)
    {
        Dg handle = Sum; // instantiate the delegate
        string str = handle(2, 5) + ":
        handle = Product;
    }
}
```
str += handle(2, 5) + "\n";
MessageBox.Show(str);
}

C# Delegate works with all primitive types including the void type. The following demonstrates how to create a delegate of void type.

using System;
using System.Windows.Forms;

class MyDel
{
    public delegate void Dg(); // declare a delegate

    public static void Sum() // delegate object
    {
        MessageBox.Show((3+7) + "\n");
    }

    static void Main(string[] args)
    {
        Dg d1 = Sum; // instantiate the delegate
        d1();
    }
}

Just like any other C# function, C# Delegates support both static and non-static functions. The following is the non-static version. It also introduces a shorthand way to create an instance of the “MyDel” class. While “m1” is an instance of the “MyDel” class, “d1” and “d2” are instances of “Dg”. By the way, “d2” delegates an anonymous instance of “MyDel” class because (new MyDel) declares an instance of “MyDel” without designating an identifier.

using System;
using System.Windows.Forms;

class MyDel
{
    public delegate void Dg(); // declare a delegate

    public void Sum() // delegate object
    {
        MessageBox.Show((3+7) + "\n");
    }

    static void Main(string[] args)
    {
        MyDel m1 = new MyDel();
        Dg d1 = m1.Sum; // instantiate the delegate
d1();

        Dg d2 = (new MyDel()).Sum; d2(); // a shorthand way
    }
}

Developers can declare the delegate without the need to pre-define its object function. C# allows developers to initialize a delegate object, known as an “anonymous” function and define the
execution statements of the function with inline code. In the following code, “d” is an instance of
the delegate and its execution is defined inline.

using System.Windows.Forms; // declare a delegate
class MyDel
{
delegate void Dg(string s);

static void Main(string[] args)
{
    Dg d = delegate(string s) // define the execution
    {
        MessageBox.Show(s);
    };

d("No predefined delegate object."); // call (invoke) d
}

A C# delegate can be initialized with a “lambda” expression. A “lambda” expression is an
anonymous function which allows you to define a parameter (if any) that can be passed with a
value for processing. A “lambda” expression can return the result of the expression. The syntax
is:

(parameters) => execution statement

The parentheses are optional only if the lambda has one input parameter; otherwise they are
required. The processing is done by execution statement defined by a pair of curly brackets next
to the lambda operator (=>). A lambda expression can then act like a function and return the
processed result through a function call. In the following, the “lambda” takes a string as an input
parameter (s). The type of s is inferred by the compiler.

using System.Windows.Forms;
class MyDel
{
delegate void Dg(string s);

static void Main(string[] args)
{
    Dg d = (s) => { MessageBox.Show(s); };

d("Lambda."); // call (invoke) d
}

The lambda expression also supports value-returning type. The following is another version of
the above code, except it returns a string type of value to its calling party.

using System.Windows.Forms;
class MyDel
{
delegate string Dg(string s);

static void Main(string[] args)
{
    Dg d = s => s;
}
Two or more parameters are separated by commas enclosed in parentheses. For example,

\[(x, y) \Rightarrow x == y\]

The complete code to demonstrate this usage is:

```csharp
using System.Windows.Forms;

class MyDel
{
    delegate bool Dg(int x, int y);

    static void Main(string[] args)
    {
        Dg d = (x, y) => x == y;
        MessageBox.Show(d(3, 2) + ""); // call (invoke) d
    }
}
```

Another sample code is:

```csharp
using System.Windows.Forms;

class MyDel
{
    delegate string Dg(string s1, string s2);

    static void Main(string[] args)
    {
        Dg d = (s1, s2) => s1 + s2;
        MessageBox.Show(d("Apple", "bee")); // call (invoke) d
    }
}
```

It is necessary to note that delegate in C# have evolved through few generations. In C#, which was the original, the delegate syntax required initialization with a named method and the use of new keyword, similar to the way to create an instance of a class. Yet, this generation is obsolete.

```csharp
delegate void Dg(string s);
delegate void M(string s)
{
    MessageBox.Show(s);
}

Dg d1 = new Dg(M);
```

**Practical Examples** Encryption is the process of converting information or data into a code, especially to prevent unauthorized access. Decryption is the reversed process. The following is the generic form of the quadratic equation.

\[f(x) = ax^2 + bx + c\]

If \(a = 3\), \(b=17\), \(c = 65\), then \(f(x) = 3x^2 + 17x + 65\).
When \( x = 671 \), then \( f(671) = 3(671)^2 + 17(671) + 65 = 1362195 \).

Let \( ax^2 + bx + c = 0 \), then the solution of the equation is expressed as:

\[
x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
\]

By letting \( 3x^2 + 17x + 65 = 1362195 \), the equation can be written as \( 3x^2 + 17x + (65-1362195) = 0 \), or \( 3x^2 + 17x -1362130 = 0 \). Therefore,

\[
x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} = \frac{-17+\sqrt{17^2-4*3*(-1362130)}}{2*3} = 671
\]

and

\[
x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} = \frac{-17-\sqrt{17^2-4*3*(-1362130)}}{2*3} = -676.6666667
\]

Apparently, it is possible to use a given quadratic equation, such as \( f(x) = 3x^2 + 17x + 65 \), for encryption and decryption. The following code, for example, is a C# function that can perform the calculation based on a given quadratic equation.

```csharp
public static int encrypt(int a, int b, int c, int x)
{
    return a*x*x + b*x + c;
}
```

The following is a C# function that can solve the given quadratic equation in the form of \( ax^2 + bx + c = e \), or \( ax^2 + bx + (c-e) = 0 \) with a premise that let \( c = c - e \). For the sake of encryption and decryption, the instructor only uses positive values for \( x \), which means \( x \geq 0 \) is the required condition and \( x < 0 \) is the rejected one. Since the solution equation will always produce two possible values, \( x_1 \) and \( x_2 \). The instructor uses an if..else statement to return only the positive value. When \( x_1 \) is positive, return \( x_1 \). When \( x_2 \) is positive, return \( x_2 \).

```csharp
public static int decrypt(int a, int b, int c, int e)
{
    c=c - e;
    double x1 = (-b + Math.Sqrt(b*b - 4*a*c)) / (2*a);
    double x2 = (-b + Math.Sqrt(b*b - 4*a*c)) / (2*a);
    if (x1 >= 0) { return (int) x1; }
    else { return (int) x2; }
}
```

The following table assigns a number to each of the uppercase alphabet.

<table>
<thead>
<tr>
<th>Letter</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>925</td>
<td>731</td>
<td>407</td>
<td>184</td>
<td>227</td>
<td>609</td>
<td>315</td>
<td>837</td>
<td>704</td>
<td>623</td>
<td>358</td>
<td>128</td>
<td>433</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Letter</th>
<th>N</th>
<th>O</th>
<th>P</th>
<th>Q</th>
<th>R</th>
<th>S</th>
<th>T</th>
<th>U</th>
<th>V</th>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>765</td>
<td>248</td>
<td>657</td>
<td>762</td>
<td>884</td>
<td>671</td>
<td>843</td>
<td>497</td>
<td>515</td>
<td>206</td>
<td>143</td>
<td>548</td>
<td>277</td>
</tr>
</tbody>
</table>

To encrypt a message, “REDWOOD”, each of the 7 alphabets will be converted to their assigned number. For example, “R” is 884, “E” is 227, “D” is 184, etc. Then, pass these numbers to the “Encrypt()” function to produce the ciphertext. For example, “R” is \( 3(884)^2 + 17(884) + 65 = 2359461 \) when \( a=3, b=17, \) and \( c=65 \). The following is a sample function that can perform the conversion from letter to number. When a character is passed to the “getX()” function, it will
find the matched case and return the associated number. By the way, a completed code is
available later.

```csharp
public static int getX(char c)
{
    int x = 0;

    switch(c)
    {
        case 'A': x = 925; break;
        case 'B': x = 731; break;
        ...........
        case 'Z': x = 277; break;
    }

    return x;
}
```

The following is the complete code for encryption. In C#, a string is defined as an array of
caracters. Therefore, “REDWOOD” is an array that has 7 elements with ‘R’ being the first
element. In the following, “REDWOOD” is assigned to a string variable named “str”, so str[0] is
to the “getX(i)” function to be converted to 884, 227, 184, and so on. These numbers are then
passed to the “encrypt()” function along with 3, 17, and 65 to be converted to a number: 21631,
15521, ..., and 15093. All the returned number are appended to the “cipher” variable.

```csharp
using System;
using System.Windows.Forms;

class Example
{
    public static int encrypt(int a, int b, int c, int x)
    {
        return a*x*x + b*x + c;
    }

    public static int getX(char c)
    {
        int x = 0;

        switch(c)
        {
            case 'A': x = 925; break;
            case 'B': x = 731; break;
            case 'C': x = 407; break;
            case 'D': x = 184; break;
            case 'E': x = 227; break;
            case 'F': x = 609; break;
            case 'G': x = 315; break;
            case 'H': x = 837; break;
            case 'I': x = 704; break;
            case 'J': x = 623; break;
            case 'K': x = 358; break;
            case 'L': x = 128; break;
            case 'M': x = 433; break;
            case 'N': x = 765; break;
            case 'O': x = 248; break;
            case 'P': x = 657; break;
            case 'Q': x = 762; break;
            case 'R': x = 884; break;
            case 'S': x = 671; break;
```
case 'T': x = 843; break;
case 'U': x = 497; break;
case 'V': x = 515; break;
case 'W': x = 206; break;
case 'X': x = 143; break;
case 'Y': x = 548; break;
case 'Z': x = 277; break;
}

return x;
}

public static void Main()
{
    string str = "REDWOOD";
    string cipher = "";
    int x = 0;
    for (int i=0; i<str.Length; i++)
    {
        x = getX(str[i]);
        if (i == str.Length - 1)
        {
            cipher += encrypt(3, 17, 65, x) + "";
        }
        else
        {
            cipher += encrypt(3, 17, 65, x) + ",";
        }
    }
    MessageBox.Show("Ciphertext: " + cipher);
}

The following is a sample output.

Although the “decrypt()” function, as discussed previously, can solve the \( f(x) = 3x^2 + 17x + 65 \) problem, it only returns numbers such as 884, 227, and 184. The instructor creates the following “getChar()” function to further convert these numbers to their associated characters. When a number is passed to the “getChar()” function, it will return the associated character (such as ‘R’).

public static char getChar(int x)
{
    char ch = 'A';
    switch (x)
    {
    case 925: ch = 'A'; break;
    case 731: ch = 'B'; break;
    .............
    case 277: ch = 'Z'; break;
    }
The following is the complete code for decryption. In C#, the **Split()** method can split a string into an array of string type using the specified delimiter. The string, "2359461,158511,,104761", will be split into an array named "e" using comma (,) as delimiter; therefore, e[0] is 2359461, e[2] is 158511, and e[6] is 104761. In the for loop, e[i] will be converted to int type using the Convert.ToInt32() method, and then passed to the "decrypt()" function along with 3, 17, 65. The "decrypt()" function will return a number like 884, 227, 184, and so on. These numbers will be immediately passed to the "getChar()" function which will return the associated character like ‘R’, ‘E’, and ‘D’. These returned letters will be appended to a string variable named “plain”.

```csharp
using System;
using System.Windows.Forms;

global class Example
{
    public static int decrypt(int a, int b, int c, int e)
    {
        c = c - e;
        double x1 = (-b + Math.Sqrt(b * b - 4 * a * c)) / (2 * a);
        double x2 = (-b + Math.Sqrt(b * b - 4 * a * c)) / (2 * a);
        if (x1 >= 0) { return (int) x1; }
        else { return (int) x2; }
    }

    public static char getChar(int x)
    {
        char ch = 'A';
        switch (x)
        {
            case 925: ch = 'A'; break;
            case 731: ch = 'B'; break;
            case 407: ch = 'C'; break;
            case 184: ch = 'D'; break;
            case 227: ch = 'E'; break;
            case 609: ch = 'F'; break;
            case 315: ch = 'G'; break;
            case 837: ch = 'H'; break;
            case 704: ch = 'I'; break;
            case 623: ch = 'J'; break;
            case 358: ch = 'K'; break;
            case 128: ch = 'L'; break;
            case 433: ch = 'M'; break;
            case 765: ch = 'N'; break;
            case 248: ch = 'O'; break;
            case 657: ch = 'P'; break;
            case 762: ch = 'Q'; break;
            case 884: ch = 'R'; break;
            case 671: ch = 'S'; break;
            case 843: ch = 'T'; break;
            case 497: ch = 'U'; break;
            case 515: ch = 'V'; break;
        }
    }
}
```
```
case 206: ch = 'W'; break;
case 143: ch = 'X'; break;
case 548: ch = 'Y'; break;
case 277: ch = 'Z'; break;
}
return ch;
}

public static void Main()
{
    string str = "2359461,158511,104761,130875,188793,188793,104761";

    string plain = "";
    char ch = 'A';
    string[] e = str.Split(',');
    for (int i=0; i<e.Length; i++)
    {
        ch = getChar(decrypt(3, 17, 65, Convert.ToInt32(e[i])));
        plain += ch + "";
    }
    MessageBox.Show("Plaintext: " + plain);
}
```

The following is a sample output.

Review
Question

1. Which C# function will return a value to its calling party?
   A. private void getDateTime() { }
   B. private static void getDateTime() { }
   C. private string getDateTime() { }
   D. private string void getDateTime() { }

2. Given the following C# function, which is the correct way to call it?
   ```
   private void getDateTime(int y) { }
   ```
   A. getDateTime()
   B. getDateTime(2020);
   C. getDateTime(2020.12);
   D. getDateTime(2020-12-25);

3. Which is the correct way to declare a C# function that will return either true or false to its calling party?
   A. private int getDateTime() { }
   B. private char getDateTime() { }
   C. private bool getDateTime() { }
   D. private char getDateTime(string t, string f) { }

Visual C# - Penn P. Wu, PhD. 225
4. Given the following C# function, which statement must be added to make it fully functional?

```csharp
private int getDateDateTime(int x)
{
    x++;  // Add this line
}
```

A. return x;
B. string str = x + "";
C. MessageBox.Show(x + "");
D. return x + "";

5. Given the following C# function, which is the correct way to call it from the Main() method of a class named "MyChar"?

```csharp
private char c()
{
    return 'A';  // Add this line
}
```

A. MessageBox.Show(MyChar.c + "");
B. MyChar c= new MyChar(); MessageBox.Show(c + "");
C. MyChar c= new MyChar(); MessageBox.Show(c() + "");
D. MyChar c= new MyChar(); MessageBox.Show(c.c() + "");

6. Given the following C# function, which is the correct way to call it from the Main() method of a class named "MyChar"?

```csharp
private static char c()
{
    return 'A';  // Add this line
}
```

A. MessageBox.Show(MyChar.c + "");
B. MessageBox.Show(c() + "");
C. MessageBox.Show(MyChar.c() + "");
D. MyChar c= new MyChar(); MessageBox.Show(c.c() + "");

7. Which can declare a function that takes an int array as parameter and returns an double array as output in C#?

A. public static void Sqrt(int[] i) { return double[]; }
B. public static void Sqrt(int[] i, double[] b) {}
C. public static int[] Sqrt(double[] b) {}
D. public static double[] Sqrt(int[] i) {

8. Which is the shorthand way to reference a non-static function named "Apple" of a C# class named "Orange"?

A. (Orange).Apple()
B. (Orange) Apple()
C. (Orange()).Apple()
D. (new Orange()).Apple()

9. Give the following code segment, which can create a delegate instance named "d" in a C# class named "Apple" and execute the Run() function?

```csharp
public delegate void Dg();
```
public void Run()
{
    MessageBox.Show("Run for life!");
}

A. Apple a = new Apple(); Dg d = a.Run; d();
B. Dg d = Apple.Run; d();
C. Dg d = (Apple) Run; d();
D. Dg d = (Apple).Run; d();

10. Given the following, which is the correct way to create a delegate with inline definition of its execution statements?

    delegate void Dg(string s);

A. Dg d = { MessageBox.Show(s); }
B. Dg d(string s) { MessageBox.Show(s); }
C. Dg d = delegate(string s) { MessageBox.Show(s); }
D. Dg d = new delegate(string s) { MessageBox.Show(s); }
Lab #7 User-Defined Functions and Delegates

Learning Activity #1: static vs. non-static functions
1. Create a new directory called C:\CIS218 if it does not exist.

2. Launch the Development Command Prompt (not the Windows Command Prompt). (See Lab #1 for details)

3. Under the C:\cis218 directory, use Notepad to create a new source file called lab7_1.cs with the following contents:

```csharp
using System;
using System.Windows.Forms;

class MyFunc
{
    static DateTime dt; // declare as class member
    static void Main()
    {
        string str = "";
        MyFunc m = new MyFunc();
        str += "Local time: " + m.getDateTime1() + "\n"; // calling non-static function
        str += "UTC time: " + getDateTime2() + "\n"; // calling static function
        m.setDate1(); // calling non-static
        str += dt.ToShortDateString() + " is a " + dt.DayOfWeek + "\n";
        setDate2(); // calling static
        str += dt.ToShortDateString() + " is a " + dt.DayOfWeek + "\n";
        MessageBox.Show(str);
    }
    private string getDateTime1() // non-static
    {
        return Convert.ToString(DateTime.Now);
    }
    private static string getDateTime2() // static
    {
        return Convert.ToString(DateTime.UtcNow);
    }
    private void setDate1() // non-static
    {
        dt = new DateTime(2019, 12, 25); // Dec 25, 2019
    }
    private static void setDate2() // static
    {
        dt = new DateTime(2020, 12, 25); // Dec 25, 2020
    }
}
```
4. Compile and test the program. A sample output looks:

5. Download the “assignment template”, and rename it to lab7.doc if necessary. Capture a screen shot similar to the above and paste it to the Word document named lab7.doc (or .docx).

**Learning Activity #2: Functions of different return types**

1. Under the C:\cis218 directory, use Notepad to create a new source file called lab7_2.cs with the following contents:

```csharp
using System;
using System.Windows.Forms;

class MyFunc
{
    static DateTime dt = DateTime.Now;
    static DateTime ny;
    
    static void Main()
    {
        MyFunc m = new MyFunc();

        string str = "Current year: " + m.Year();
        str += "Leap year: " + m.leap();
        str += "AM/PM: " + m.Ampm() + "m";
        m.setDateTime();
        str += "Day(s)/time to next Jan 1: " + (ny - dt); //output format day.H:M:S.ms
        str += "Day(s) to next Jan 1: " + m.showDayOnly();
        MessageBox.Show(str);
    }

    private int Year()
    {
        return dt.Year; // get current year value
    }

    private bool leap()
    {
        return DateTime.IsLeapYear(dt.Year);
    }

    private char Ampm()
    {
        return (dt.Hour > 12 ? 'p' : 'a'); // PM starts at 12:00
    }

    private void setDateTime() // return string
    {
        int y = dt.Year; // current year
        ny = new DateTime((y+1), 1, 1, 0, 0, 0); // 00:00:00 of Jan 1 of next year
    }
```
private string showDayOnly()
{
    string d = Convert.ToString(ny - dt);
    string s = "";

    for (int i=0; i<d.Length; i++) // d is string and a char array by nature
    {
        if (d[i]=='.') { break; } // terminate the loop if the first dot appears
        else
        {
            s += d[i]; // append char to s
        }
    }

    return s;
}

2. Compile and test the program. A sample output looks:

3. Capture a screen shot similar to the above and paste it to the Word document named lab7.doc (or .docx).

Learning Activity #3: array as return type – SuperLotto simulation

1. Under the C:\cisc218 directory, use Notepad to create a new source file called lab7_3.cs with the following contents:

```csharp
using System;
using System.Windows.Forms;

public class Example
{
    static Random rn = new Random();

    public static int[] getIt(int[] a) // return an int array
    {
        int temp;
        int x;

        for (int i=46; i>=0; i--) // a shuffle algorithm
        {
            x = rn.Next(1, 47); // randomly pick a number for x
            temp = a[i]; // assign the value of the last element to temp
            a[i] = a[x]; // swap the value of a[x] and a[last]
            a[x] = temp;
        }

        return a;
    }

    public static string showIt(int[] a) // take array return string
    {
        string str = "";
```
for (int i=0; i<5; i++) // get first 5 values
{
}
return str;
}

static void Main()
{
    int[] n = new int[47];

for (int i=0; i<47; i++) // generate 47 numbers
{
    n[i] = i + 1; // set the value to n[i] + 1 and assign it to n[i]
}

int d = Convert.ToInt32(InputBox.Show("How many drawing?");

string str = "California SuperLotto:\n";
for (int i=0; i<d; i++)
{
    int[] m = n;
    getIt(m);
    str += showIt(m) + " mega: " + rn.Next(1, 28) + "\n";
}

MessageBox.Show(str);
}

2. Compile and test the program. A sample output looks:

3. Capture a screen shot similar to the above and paste it to the Word document named lab7.doc (or .docx).

Learning Activity #4: Parameters and Reusability

1. Under the C:\cis218 directory, use Notepad to create a new source file called lab7_4.cs with the following contents:

```csharp
using System;
using System.Windows.Forms;

class MyFunc
{
    static DateTime ny;
    static string str;

    static void Main()
    {
        str = "Day of Week: 2020 ~ 2029\n\n";
        str += "Year\tNew Year\tJuly 4th\tChristmas\tFeb 29\n";
```
for (int i=2020; i<2030; i++) // call functions to reuse them 10 times
{
    str += i + "\t"; // tab
    findNewYear(i);
    str += findIndept(i, 7, 4).DayOfWeek + "\t"; // Jul 4, YYYY
    str += findCmas(new DateTime(i, 12, 25)) + "\t"; // Dec, 25, YYYY
    str += Leap(i, DateTime.IsLeapYear(i)) + "\n"; // Feb 28, YYYY
}
MessageBox.Show(str);
}

public static string findCmas(DateTime dt) // dt is parameter
{
    return Convert.ToString(dt.DayOfWeek);
}

public static void findNewYear(int y) // y is parameter
{
    ny = new DateTime(y, 1, 1); // Jan 1, YYYY
    str += Convert.ToString(ny.DayOfWeek) + "\t";
}

public static DateTime findIndept(int y, int m, int d) // multiple parameter
{
    return (new DateTime(y, m, d));
}

public static string Leap(int y, bool b) // parameter of different types
{
    if (b)
    {
        return Convert.ToString(new DateTime(y, 2, 19).DayOfWeek);
    }
    else
    {
        return "N/A";
    }
}

2. Compile and test the program. A sample output looks:

3. Capture a screen shot similar to the above and paste it to the Word document named lab7.doc (or .docx).

Learning Activity #5: Concept of delegate
1. Under the C:\cis218 directory, use Notepad to create a new source file called lab7_5.cs with the following contents:

```csharp
using System;
using System.Windows.Forms;

class MyDel
{
    public delegate int Dg(int x, int y); // declare a delegate

    public static int Sum(int x, int y) // delegate object
    {
        return (x + y);
    }

    public static int Difference(int x, int y)
    {
        return (x - y);
    }

    public static int Product(int x, int y)
    {
        return (x * y);
    }

    public static int Remainder(int x, int y)
    {
        return (x % y);
    }

    static void Main(string[] args)
    {
        Dg handle = Sum; // instantiate the delegate
        string str = handle(2, 5) + "\n";

        handle = Difference;
        str += handle(2, 5) + "\n";

        handle = Product;
        str += handle(2, 5) + "\n";

        handle = Remainder;
        str += handle(2, 5) + "\n";

        MessageBox.Show(str);
    }
}
```

2. Compile and test the program. A sample output looks:

3. Capture a screen shot similar to the above and paste it to the Word document named lab7.doc (or .docx).

**Submittal**

1. Complete all the 5 learning activities.
2. Create a .zip file named lab7.zip containing ONLY the following self-executable files.
   • Lab7_1.exe
   • Lab7_2.exe
   • Lab7_3.exe
   • Lab7_4.exe
   • Lab7_5.exe
   • Lab7.doc (or .docx) [You may be given zero point if this Word document is missing]

3. Log in to course site and enter the course site.

4. Upload the zipped file as response to question 11 of Assignment.

**Programming Exercise:**
1. Use Notepad to create a new file named ex07.cs with the following heading lines (be sure to replace YourFullNameHere with the correct one):

   ```csharp
   //File Name: ex07.cs
   //Programmer: YourFullNameHere
   ```

2. Under the above two heading lines, write C# codes to create a function named “findWDay()” of the string type. Make the “findWDay()” function take one parameter of DateTime type. Inside the “findWDay()” function, use the DayOfWeek property of the DateTime structure to return the “day of the week” value as string to the calling party. In the Main() method uses a message box to call the “findWDay()” function, pass DateTime values in a correct format to get the “day of the week” value of December 31 for the next 5 years to. If the current year is 2020, make 2021 the first year, 2022 the second, and so on. Display the returned value similar to the following on the message box.

```
Year    Dec 31
2020 Thu
2021 Fri
2022 Sat
2023 Sun
2024 Tue
``` 

3. Download the “programming exercise template”, and rename it to ex07.doc. Capture a screen shot similar to the above figure and then paste it to the Word document named “ex07.doc” (or .docx).

4. Compress the source code (ex07.cs), the executable (ex07.exe), and the Word document (ex07.doc or .docx) to a .zip file named “ex07.zip”. You may be given zero point if any of the required file is missing.

**Grading Criteria:**
- You must be the sole author of the codes.
- You must meet all the requirements in order to earn credits.
- No partial credit is given.