Creating a console application with Visual Studio .NET

This lab will familiarize you with the Visual Studio IDE and various types provided by the .NET framework. After this lab you should understand something about:

- Source code files
- The solution explorer
- The properties window
- The locals window
- String formatting
- The System Namespace:
    - Static methods of the Console class
    - Static methods of the char type
- The System.IO Namespace
    - Static method of the File class
- .NET Types such as stirng, DateTime, char, int
- How to read from the standard input
- How to write to the standard output
- Importing namespaces with the "using" keyword

Building a console application with Visual Studio .NET

Exercise 1
Building a console application

Scenario

Create a console application, and discover Visual Studio.NET

Tasks	Detailed Steps
1. Setting up the lab	**a.** Open Visual Studio and create a new C# console application project named ConsoleApplication1. **Note:** For convenience, you should create the project on your flash drive. **b.** Notice that the project has created one source file named Program1.cs. Double-click on Program.cs file to see the source code. **c.** Notice the "using" statements at the top of the file. This statement makes types defined within the listed namespace available. For example, if "using System;" is provided, you can reference the Console class directly instead of fully qualifying it as System.Console.

2. Use the write methods of the Console class	**a.** The Console class represents the standard input and standard output for console applications. Using this class, you can write text to the screen or read user input from the keyboard.
	b. In the Main method, write the text "Hello World " followed by the current date and time.
	
	Note: The current date/time is obtained by calling the "Now" property of the DateTime class.
	Be sure to put a Console.ReadLine() statement at the end of the Main method. This will keep the command window open when invoked from Windows.
	
	<pre><code>static void Main(string[] args)
	{
	    Console.Write("Hello World ");
	    Console.Write(DateTime.Now.ToString());
	
	    Console.ReadLine(); //this keeps the console window open
	}</code></pre>**c.** Run your application and notice the output.

3.	Adding line breaks	**a.** You can add a line break after text by using the WriteLine method. To print our "Hello World" text on two lines we would use the WriteLine method to print "Hello":
		`Console.WriteLine("Hello");`
		`Console.Write("World");`
		b. You can also add a line break inline using \n:
		`Console.Write("Hello\nWorld");`

4.	Use the Read methods of the Console class	**a.** To accept input from the user, use the Read or ReadLine method of the console class. These methods return a string.
		b. The read method will return one character at a time while the ReadLine method returns all characters up to when the user hits the enter key.
		c. In the Main() method, Create a variable (field in OO parlance) of type string named "name":
		`string name;`
		d. Call the ReadLine method of the Console class and assign the result the the "name" variable:
		`name = Console.ReadLine();`
		e. In the Main method, put a breakpoint on the line where you declared the "name" variable. You add a breakpoing by clicking the left grey margin of your source code window.
		f. Run your application. When your executing code hits the breakpoint, execution will halt and you will be able to inspect the state of your running application.
		g. Once your application halts, open the "Locals" window (Ctrl+D,L) and notice that the "name" field is empty. Hit F11 to step thorugh the code. Once you've provided some input, notice the value is assigned to the "name" field.

```
    using System;

  namespace ConsoleApplication1
  {
      class Program
      {
          static void Main(string[] args)
          {
              Console.Write("Hello World \n");
              Console.Write(DateTime.Now.ToString());

              string name;
              name = Console.ReadLine();

              Console.ReadLine();
          }
      }
  }
```

```
Locals                                                              ☒
  Name              Value                        Type              ▲
  ● args            {string[0]}                  string[]
  ● name            "keith"              🔍 ▾    string



                                                                  ▼
  ↩ Undo Close  🔲 Locals  🔲 Watch 1
```

Note: You can declare a field and set it value on the same line:
```
string name = Console.ReadLine();
```

5. Formatting strings

a. While you can concatinate strings together using the + operator, the Write methods of the console class make this easier by supporting placeholders as replaceable parametes. For example, consider the following code:

```
Console.Write("Enter first name: ");
string fname = Console.ReadLine();
Console.Write("Enter last name: ");
string lname = Console.ReadLine();

Console.WriteLine("Your first name is {0} and your last
name is {1}", fname, lname);
```

b. The placeholder {0} is replaced with the first filed, fname and {1} Is replaced with the second parameter, lname. Remember that indexes in C# always start with 0.

c. Now wite your own application that prompts the user to enter his/her username and password and then prints the following output:

```
<PLEASE LOG IN>

Enter your username: keith
Enter your password: foobar

User logged in at 6/10/2008 4:24:40 PM.
Login credentials:
        username: keith
        password: foobar

---- press any key to end ----
```

Note: Inside a string literal, certain non printable characters are represented as escape sequences :

```
  \n – print a new line
  \t – print a horizontal tab

Other reserved characters must also be escaped:
  \" – print a double quote
  \\ - print a backslach character
```

Please see http://msdn.microsoft.com/en-us/library/ms228362.aspx for more information.

| 6. Converting strings | a. The Read methods of the Console class only return strings. If you need to accept a user entered string that represents a different type such as an integer or a date, you must convert it. Remember that C# is a type-safe language that will not try to infer a value. |

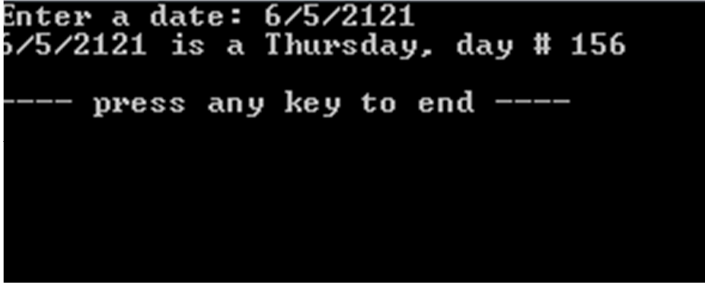b. To cast a string to another type, use the methods of the Convert class.

c. Enter this code into the Main method of a console class:

```csharp
Console.WriteLine("This program will add 2 numbers.");
Console.Write("Enter the first number: ");
string num1 = Console.ReadLine();
Console.Write("Enter the second number: ");
string num2 = Console.ReadLine();

int n1 = Convert.ToInt32(num1);
int n2 = Convert.ToInt32(num2);

Console.WriteLine("{0} + {1} = {2}", num1, num2, n1+n2);

Console.WriteLine("\n---- press any key to end ----");
Console.ReadLine();
```

d. Using Visual Studio.NET you can inspect any type. This is available because .NET uses metadata when describing types and is enabled through "reflection". Right-click on the Convert class and choose "Go To Definition". The Convert class will open in a new window. Notice the other ToXXX methods that exist.

e. Now write a console application that will accept a date from the user and return the day of the week.


```
Enter a date: 6/5/2121
6/5/2121 is a Thursday, day # 156

---- press any key to end ----
```

Note: fields of type DateTime contain methods that give the day of the week and year.

BONUS ROUND

1. Write an application to read a text file and generate a report on the number of words, letters, punctuation marks, numbers, and whitespace characters it contains.
2. To get started, load the Session1.Lab1.NYU project and write code as indicated by the "todo:" comments.
3. If you get stuck, show all files in the solution explorer and look at the Program_Solution.cs file.

Note: the char type contains some static methods that are helpful in completing this exercise.

Example:

```
if (char.IsNumber(c))
    num chars++;
```

Note:
In the solution explorer window, click on the lipsum.txt file and notice in the properties window that "Copy to Build Directory" value is set to "Copy Always". This means that on compilation, the lipsum.txt file will be copied to the bin directory along with the .exe file. Because the text file is in the same directory as the executable, you call File.ReadAllText("lipsum.txt") without having to pass the

	`full path to the file.`