C# Programming Tutorial
Lesson 1: Introduction to Programming

About this tutorial

This tutorial will teach you the basics of programming and the basics of the C# programming language.

If you are an absolute beginner this tutorial is suited for you.

If you already know one or more programming languages, you might find it a bit boring and skip to the next lesson.

To follow this tutorial you need to have Visual C# Express Edition 2008 or 2010 installed on your computer. These applications are free to download and install.

The best way to learn this is by practicing. Make sure you write all the examples yourself and test them, and that you do the tasks that I have put at the end. The tasks at the end will probably help you the most to get used to C#.

This tutorial has been entirely created by Davide Vitelaru (http://davidevitelaru.com/).

Note: You can use the table of contents at page 20 to get around the document quickly

Software required:
- Visual C# Express Edition 2008/2010

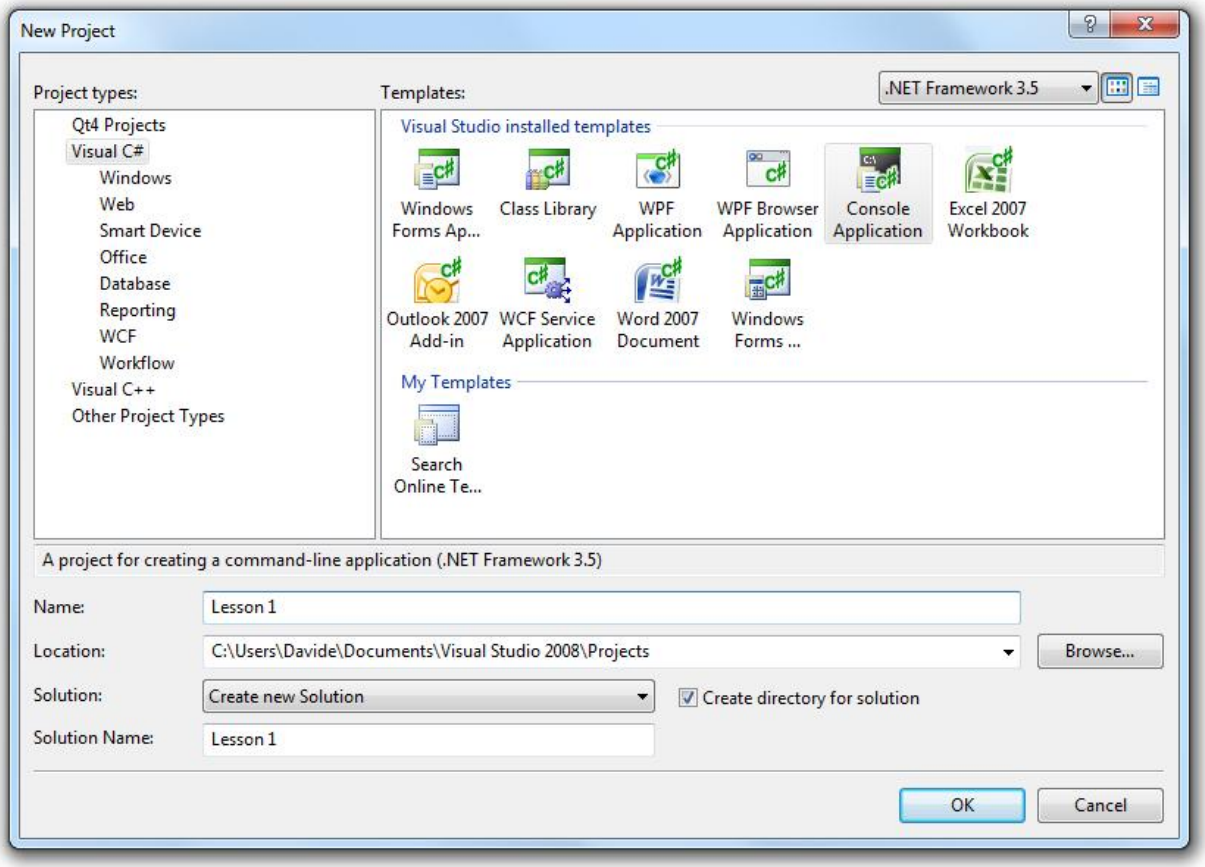
You must know:
- What programming is
- What a programming language is

You will learn:
- Some Basics
- Variables
- Variable Operations
- Decisions
- Loops

Some Basics

Throughout this tutorial I will refer to Visual C# Express 2008/2010 as the IDE (Integrated Development Editor).

To start with, open your IDE and create a new project (File >> New >> Project or Ctrl + Shift + N). Select the **Visual C# Console Application** template from the window that appears and click OK:



Once you created your project, you will see this:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Lesson_1
{
    class Program
    {
        static void Main(string[] args)
        {

        }
    }
```

```
}
```

I know it looks scary, but it's not that complicated. You only have to worry about this section:

```
        static void Main(string[] args)
        {

        }
```

This is the exact place where you will write your source code, to be exact, between the braces following `static void Main(string[] args)`.

At this point, your application won't do anything. To start you application, press F5. You will see a black windows appearing and closing immediately.

It closes immediately because it does exactly what you told it to do: nothing. Let's "tell" it to open and wait for a keystroke to close.

Write the following line between the braces of `static void Main(string[] args)`:

```
            Console.ReadKey();
```

Now, press F5 to run your application. You will end up with a black window awaiting you to press any key so it closes.

Let's make it even more fun, make your code look like this:

```
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            Console.ReadKey();
        }
```

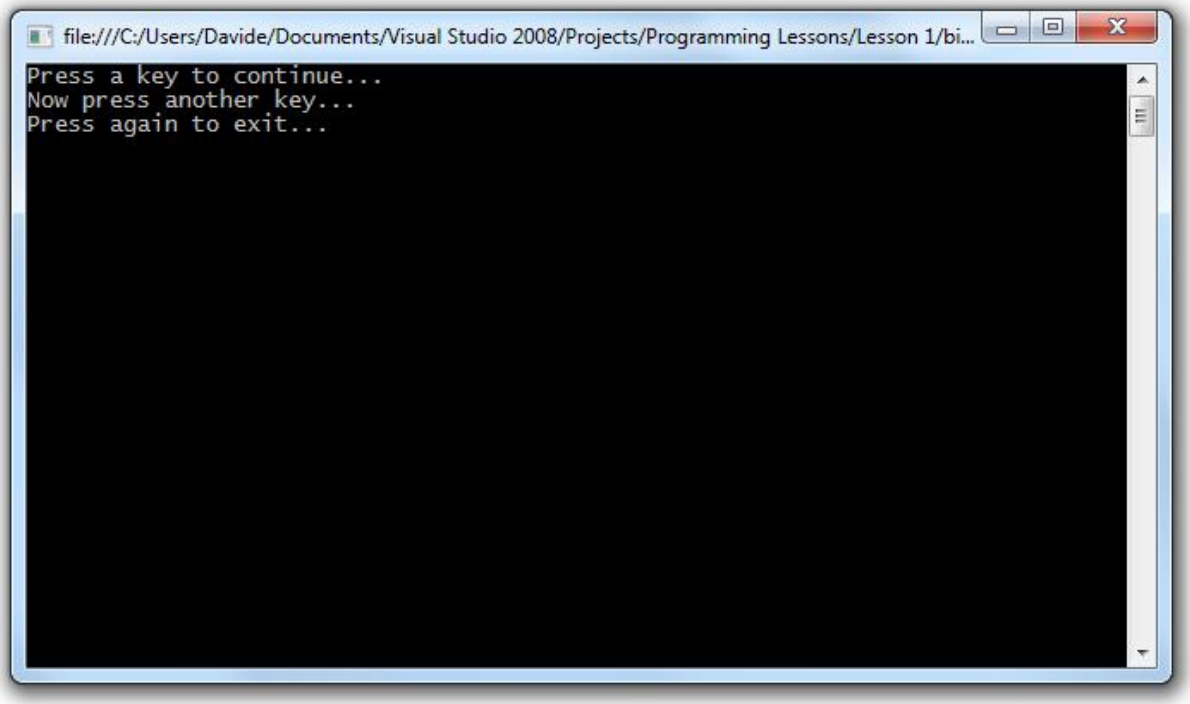
Again, press F5 to run your application. This time the application will display "Hello World" and then it will wait for you to press a key. If you get an error, make sure you typed everything correctly. Also, don't forget the semicolons at the end; they are very important (and annoying for beginners that keep forgetting them).

A statement can be used multiple times. Do the following:

```
        static void Main(string[] args)
        {
            Console.WriteLine("Press a key to continue...");
            Console.ReadKey();
            Console.WriteLine("Now press another key...");
            Console.ReadKey();
```

```
            Console.WriteLine("Press again to exit...");
            Console.ReadKey();
        }
```

Just change the text between the quotation marks in the `Console.WriteLine("")` statement to change the displayed message.



What's the catch with the black window?

The black window that you are currently working at is called a console window. Back in the 1980's computers didn't have taskbars and windows like they do now, the only had this text-based interface. Your application has a text-based interface at the moment.

Creating an application with a user interface (windows, buttons, text boxes, etc…) is usually harder, but thanks to Microsoft's .NET framework we can create one in a few easy steps; yet, that is not the point of this lesson.

This lesson is supposed to show you the basics, and once you finish it you will be able to move on to further lessons and create useful and good-looking applications.

Data manipulation

A program that displays messages and waits for keystrokes won't be of use to anyone, so let's make it do something useful. Let's make it add two numbers.

Variables

Variables are like boxes, you can put things in them. In our case, we will use them to store values.

Variables are of different types, depending on the type it can store different values, for example and **integer** variable can hold a number, while a **string** can hold characters (ex. "hello my name is john" – 21 characters, spaces included).

To start with, let's use variables display information:

```
        static void Main(string[] args)
        {
            string name;
            name = "John";
            Console.WriteLine(name);
        }
```

Press F5, run your application and see the result. If you receive an error, make sure you typed everything correctly.

How does it work?

To use a variable, we must first create it. To create it (a better term would be to "declare" it), you must type the variable type, followed by the name you want the variable to have:

```
            string variable;
            int another_variable;
```

At this point, both of these variables are empty. To assign a value to a variable, type the name of the variable, equal and the value you want it to hold. If it is a string, never forget to type the value between quotation marks:

```
            variable = "hello there";
            another_variable = 22;
```

Make sure you assign the correct type of value to the variable, or you will receive an error; In this case `variable` is a `string` so it can hold a string value, and is `another_variable` an `int`eger so it can hold a number. You can name the variables however you like as long as you don't use reserved words (like `int`, you can't do `int int` because it would return an error), and the name doesn't contain some particular symbols, and the name doesn't start with a number.

Let's make the computer ask for our name, and then greet us:

```
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, what is your name?");
            string name;
            name = Console.ReadLine();
            Console.WriteLine("Hello, " + name);
            Console.ReadKey();
        }
```

What this code does is to declare a variable called `name` and then to assign it the value of the user's input.

Press F5 and introduce yourself to your program.

How does it work?

`Console.ReadLine()` represents the users input, or what you type in the console window. You can assign that input to a variable as seen in the example above.

You can also tie together two strings using the + sign; in this case we tied together `"Hello"` and the variable `name` which is a string too. You can also do `"hello " + "there"` and get "`hello there`".

Making a calculator would seem to be pretty easy, and it is, but you have to remember one thing: the user input is a string; therefore you cannot assign it to an integer unless you convert it.

```
        static void Main(string[] args)
        {
            int number1, number2;

            number1 = Int32.Parse(Console.ReadLine());
            number2 = Int32.Parse(Console.ReadLine());

            Console.WriteLine(number1.ToString() + "+" +
number2.ToString() + "=" + (number1 + number2).ToString());

            Console.ReadKey();
        }
```

Press F5, and make sure you type a number and press enter, then type another number and press enter then stare at the result before pressing a key to exit. If you type anything but numbers it will return you an error so be careful.

Note that you can declare variables of the same type by separating the names with a comma (`int number1, number2`).

Int32.Parse() will turn what you put between the parenthesis into an integer, as long as it's a string.

Example: `int x; x = Int32.Parse("20");`

In the previous example we used `Int32.Parse()` to convert the user input (that is a string) into an integer and assign it to two integer variables.

Since the two variables (`number1, number2`) are integers, you can't just display them without converting them to strings. To convert them, just type the integer's name followed by "`.ToString()`". This will convert any integer variable, or sum of an integer variable into a string.

As you can see, we used a parenthesis to convert only the sum of the two variables:

`(number1 + number2).ToString()` – this will convert the sum of number1 and number2 into a string.

`Number1.ToString() + number2.ToString()` – this will only convert them separately and tie them together (Ex. `"2" + "2" = "22"`).

Variable Operations

You already know how to create a calculator that can add, but let's also make it subtract, multiply and divide.

Small side note: if you type // in your C# source code, all that remains of the line will be turned into a comment. The comment has no importance for the application, but only for the programmer. I will use comments in the source code to explain things easier. Comments are colored in green in the source code.

```
        static void Main(string[] args)
        {
            //We declare two integers
            int number1, number2;

            //We ask the user for values
            Console.WriteLine("Please insert a number:");
            number1 = Int32.Parse(Console.ReadLine());
            Console.WriteLine("Please insert another number:");
            number2 = Int32.Parse(Console.ReadLine());

            //We create a variable to hold the results
            //and use it in calculations
            int result;

            result = number1 + number2;
            //Addition, use + to add two integers
            Console.WriteLine("Sum: " + result.ToString());

            result = number1 - number2;
            //Subtraction, use - to subtract two integers
            Console.WriteLine("Subtraction: " + result.ToString());
```

```
            result = number1 * number2; //Multiplication, use *
            Console.WriteLine("Multiplication: " + result.ToString());

            result = number1 / number2; //Division, use /
            Console.WriteLine("Division: " + result.ToString());

            Console.ReadKey();
        }
```

Press F5 and try it out.

Decisions

Sometimes, you will have to execute just a piece of code depending on the user's input. For example, if the user has inserted a number and you want your application to display if the number is positive or negative, you will need some extra pieces of code.

```
        static void Main(string[] args)
        {
            Console.WriteLine("Insert a number: ");
            //As you can see, a variable can be
            //assigned while it is declared (created)
            int number = Int32.Parse(Console.ReadLine());

            if (number > 0) Console.WriteLine("Number is positive");
            else if (number == 0) Console.WriteLine("Number is 0");
            else Console.WriteLine("Number is negative");

            Console.ReadKey();
        }
```

This code is easy to understand, one of the advantages of C# being the face that it's similar to English.

If the number is greater than 0, display that the number is positive, else if the number is 0 display that it is 0, else display that it is negative.

The equality operator is == because = is used to assign. The following statement would return an error:
`if (number = 0) Console.WriteLine("Number is 0");`

What do we do when we want to do multiple things under the same `if` clause?

```
            if (number > 0)
                Console.WriteLine("Greater than 0");
            number = 0;
```

This would assign 0 no matter what number is, but if we insert both statements between braces following the "`if`" clause we might get lucky:

```
            if (number > 0)
            {
                Console.WriteLine("Greater than 0");
                number = 0;
            }
```

Now, if the number is greater than 0, it will be assign the value 0 after the message is displayed.

This is how braces are used to execute multiple statements.

Let's make a calculator that lets the user decide what operation to perform.

Try to do it yourself, it would be good practice, then look at the code. Small tip: for strings just do `if (variable == "addition")`, it's the same syntax as it is for integers.

```
        static void Main(string[] args)
        {            Console.WriteLine("Please insert two numbers:");
            int n1 = Int32.Parse(Console.ReadLine());
            int n2 = Int32.Parse(Console.ReadLine());

            Console.WriteLine("Type operation to perform:");
            string decision = Console.ReadLine();

            if (decision == "Add")
                Console.WriteLine((n1 + n2).ToString());
            else if (decision == "Subtract")
            {
                int result;
                result = n1 - n2;
                Console.WriteLine(result.ToString());
            }
            else if (decision == "Multiply")
            {
                int result = n1 * n2;
                Console.WriteLine(result.ToString());
            }
            else if (decision == "Divide")
            {
                string result = (n1 / n2).ToString();
                Console.WriteLine(result);
            }
            else Console.WriteLine("Invalid choice");

            Console.ReadKey();
        }
```

Note that I used different ways to calculate the result just to show you how you how flexible the variable operations are.

Of course, you can use an `if` inside another:

```
            if (n1 > 0)
            {
                if (n1 > 10)
                    Console.WriteLine("Greater than 10");
                else Console.WriteLine("Smaller than 10");

                Console.WriteLine("Greater than 0");
            }
```

What if we want to check if a variable does NOT hold a value?

```
            //If the name is not John, display
            //Hello, else display Hello John
            if (name != "John")
                Console.WriteLine("Hello!");
            else
                Console.WriteLine("Hello John!");
```

Loops

Where would we be without loops? Imagine that we would have to write the same statement all over again, and it might still not work.

Just imagine asking the user for a question, if he gets it wrong, what would the application do? Exit so he can start all over, or repeat the question?

There are several ways you can make your application repeat a statement.

While loop

This loop will repeat a statement until something happens.

```
        static void Main(string[] args)
        {
           //Variable x is 5
            int x = 5;

            while (x > 0)
            {
                //While x is greater than 0,
                //we display it's value and decrease it
                Console.WriteLine(x.ToString());
                x = x - 1;
            }

            Console.ReadKey();
        }
```

As always, press F5 to test your program. It will display all numbers from 5 to 1. It will not display 0 because x will be greater than 0 at that point. What you can do is use the greater or equal operator:

```
            while (x >= 0)
            {
                //While x is greater than 0,
                //we display it's value and decrease it
                Console.WriteLine(x.ToString());
                x = x - 1;
            }
```

This will also display 0. It also works for smaller or equal (<=).

The “while” loop can be used to repeat questions:

```
        static void Main(string[] args)
        {
            string password = "pass";
```

```
            while (password != "1234")
            {
                Console.WriteLine("Please insert your password:");
                password = Console.ReadLine();
            }

            Console.WriteLine("Password correct!");
            Console.ReadKey();
        }
```

This code will ask the user for the password (1234) and it will only stop when he gets it right. The last two lines of code will be executed only if the user manages to "escape" the loop, but he cannot do that unless he types the correct password. Press F5 and try it yourself.

Note that we assigned a value to the "password" variable before using it in the "while" loop. If we do not assign a value to it, it will return an error.

Also, if the condition is never accomplished, the loop will run to infinity, so you better be careful what you condition is:

```
        static void Main(string[] args)
        {
            while ((2 + 2) == 4)
            {
                Console.WriteLine("This will never stop");
            }
        }
```

For loop

The "for" loop is different from the while loop by the fact that it allows you to count the times you loop. Let's imagine you want to display "Hello!" fifty times on the screen. You could type `Console.WriteLine("Hello!")` fifty times in the source code, but that would just be wrong.

Instead, use a "for" loop:

```
        static void Main(string[] args)
        {
            //i++ is the same thing with i = i + 1
            //You can use any integer to do that
            //Example: int variable = 0; variable++;

            for (int i = 0; i <= 50; i++)
            {
                Console.WriteLine("Hello!");
            }

            Console.ReadKey();
```

```
        }
```

I know it looks scary, but it's pretty easy. What the "for" loop does it to declare a variable named "i" (you can name it anyway you want), you assign it the value 0, you "tell" it that it must not exceed 50, and you "tell" it to increase itself by 1 in each loop. Once it reaches 50, it will stop.

Press F5 and test it. **If you receive any error, make sure you didn't forget any semicolon in the "for" statement**.

Note that using ++ after an integer will increase it by 1. If you already have an integer declared, you can use it in the "for" loop instead of declaring another one.

```
        static void Main(string[] args)
        {
            int variable;

            //You don't have to use { } if you only have one statement
            for (variable = 0; variable < 10; variable = variable + 1)
                Console.WriteLine(variable.ToString());

            //As you can see, I used variable = variable + 1 instead
of variable++

            //And yes, you can display the value of the counter
variable (in this case
            //cleverly named "variable", but named "i" in the previous
example)

            Console.ReadKey();
        }
```

This example will display the numbers from 0 to 9.

Summary

If you are working on a project and forgot something, check this section always.

This is what you learned in this lesson, plus a few extra things so make sure you read this.

Simple statements

`Console.WriteLine("Hello");`	Displays on the console window the string placed between the parentheses.
`Console.ReadKey();`	Prompts the user to press any key.

Variables

Declaration: `variable_type variable_name;`

Example: `int name;`

Known types (at this point): `int, string.`

Assigning: `variable_name = value;`

Variable Operations

For integers:

```
int x;
int y;

x = Int32.Parse(Console.ReadLine());
y = Int32.Parse("5");

x = y + 5 + 10; // x will be y + 5
x = (2 + 2) + 2 / 2 + 10; //This works too
//Use as many brakets as you need
x = x + y; // x will be x + y
x++; // x will be x + 1

x += 5; //x will be x + 5
x *= 5; //x will be x * 5
//Same for / and -
x /= 5; x -= 5;
```

For strings:

```
name = "John";
age = "17";

name = "Name is:" + name + ", and the age is:" + age;
```

```
            //name will hold: "Name is: John, and the age is: 17"

            Console.WriteLine(name);

            int x = 15;

            name += x.ToString();
            //+= works here too, don't try -=, *= and /= though
            //name will hold: "Name is: John, and the age is: 1715"
```

Decisions

```
            if (x != 0) Console.WriteLine("Number is NOT 0");

            if (x == 0) Console.WriteLine("Number is 0");

            int y = 5;

            //You can use && to use two conditions
            if (x > 0 && y > 0)
                Console.WriteLine("Both numbers are positive");
            //The equivalent:
            if (x > 0) if (y > 0)
                    Console.WriteLine("Both numbers are positive");

            //This is the OR operator ||
            //If any of the conditions is acomplished,
            //the staments will be executed
            if (x > 0 || y > 0)
                Console.WriteLine("One of the numbers is positive");

            //Both operators can be used more times:
            if (x > 0 && y > 0 && (2 + 2 == 4))
            {
                Console.WriteLine("Both numbers are positive");
                Console.WriteLine("The computer agrees that 2+2 equals
4");
            }

            //If x is greater than 0 OR 1 + 1 equals 2
            if (x > 0 || (1 + 1 == 2))
            {
                //If this doesn't run then you computer can't
calculate
                Console.WriteLine("I don't know if the number is
positive");
                //This statement will run anyway because 1+1==2 is
always true
            }
```

Loops

```
            //WHILE LOOP
            string username = "", password = "";

            //While the username is NOT John
            //AND &&
            //The password is not 1234
            while (username != "John" && password != "1234")
            {
                Console.WriteLine("Username and password:");
                username = Console.ReadLine();
                password = Console.ReadLine();
            }

            Console.WriteLine("Hello John!");

            //FOR LOOP
            for (int i = 0; i < 10; i = i + 2)
            {
                //i = i + 2
                //This will increment i by two each loop

                //You can do any operation you want there
                //Multiply by 2:
                //for (int i = 0; i < 10; i = i * 2)
                //or
                //for (int i = 0; i < 10; i *= 2)

                Console.WriteLine(i.ToString());
            }
```

Tasks

As I promised, these are the tasks. The task actually since there is a single one in this lesson, and it might be pretty challenging for a beginner.

Make sure you try to do them yourself before looking at the source codes. After all, the whole idea is to learn programming.

Task – Calculator

Create a simple calculator that prompts the user for 2 numbers, and then asks the user what the operation that you want it to perform is.

The calculator must be able to do the following operations:

- Addition
- Subtraction
- Multiplication
- Division
- Exponentiation (Number multiplies by itself, if you're not 7^{th} grade yet don't do it)

At the end, the calculator must ask the user if he wants it to perform another calculation, and do so if he does.

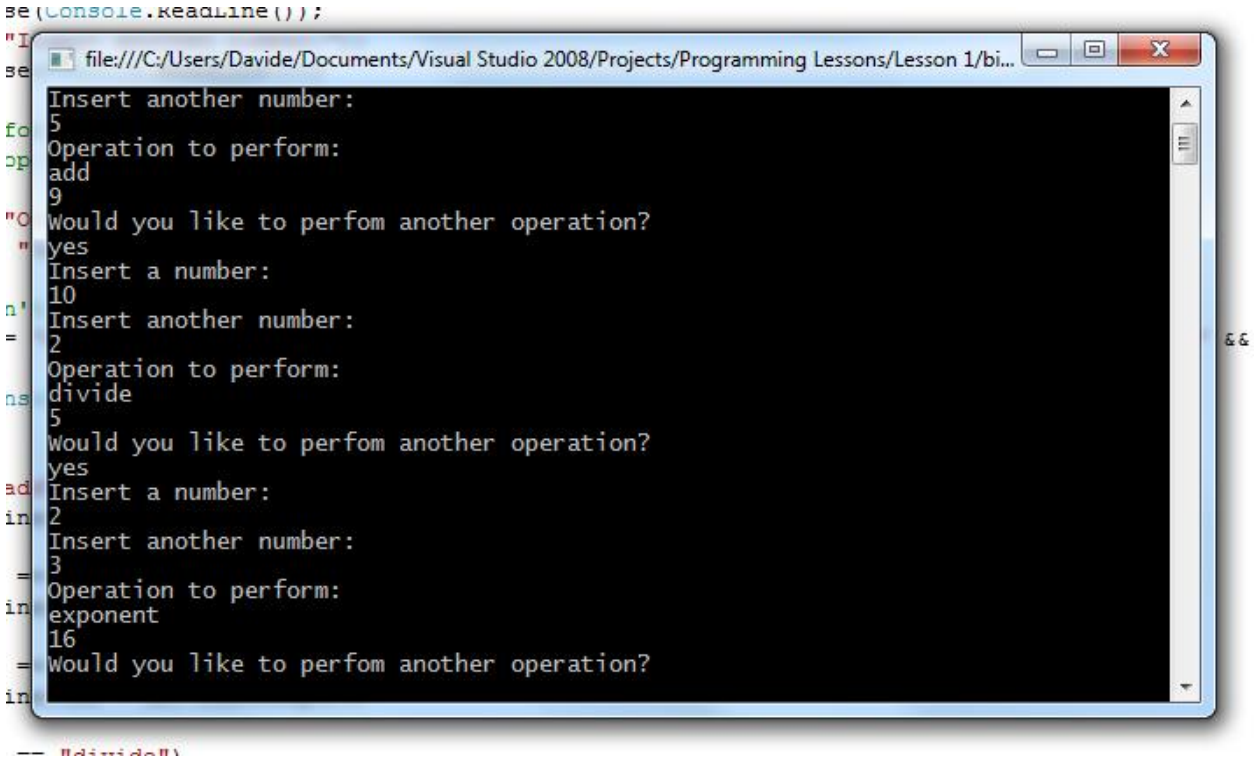
Tips: Use a "for" loop for the exponentiation, and a variable that holds the user's answer to perform another operation at the end along with a "while" loop. Also, you can write the way the application will work on a piece of paper, and then "translate" what you wrote in C#.

Code:

```
        static void Main(string[] args)
        {
            //This variable holds the user's
            //decision to perform another operation
            string answer = "yes";

            //While the answer is "yes"
            while (answer == "yes")
            {
                //Asking the user for the numbers
                Console.WriteLine("Insert a number: ");
                int n1 = Int32.Parse(Console.ReadLine());
                Console.WriteLine("Insert another number:");
                int n2 = Int32.Parse(Console.ReadLine());

                //Asking the user for the operation to perform
                //Using a while loop to make sure that he gets it
right

                Console.WriteLine("Operation to perform: ");
                string operation = "none"; //This will hold the answer

                //If the user doesn't insert a correct answer, he will
be asked to insert an answer again
                while (operation != "add" && operation != "subtract"
&& operation != "multiply" && operation != "divide" && operation !=
"exponent")
                {
                    operation = Console.ReadLine();
                }

                if (operation == "add")
                    Console.WriteLine((n1 + n2).ToString());

                else if (operation == "subtract")
                    Console.WriteLine((n1 - n2).ToString());

                else if (operation == "multiply")
                    Console.WriteLine((n1 * n2).ToString());

                else if (operation == "divide")
                    Console.WriteLine((n1 / n2).ToString());

                else if (operation == "exponent")
                {
                    int result = 1;

                    //This will multiply n1 with itself for n2 times
                    for (int i = 0; i <= n2; i++)
```

```
                        result *= n1;

                    Console.WriteLine(result.ToString());
                }

                //Asking the user for another operation
                Console.WriteLine("Would you like to perfom another
operation?");
                answer = Console.ReadLine();
            }
        }
```

Watch it running:



Contents