First C# program

Here is the code for our first C# program. Let's look at it line by line.

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

namespace FirstCSharp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter your Name");
            string name = Console.ReadLine();
            Console.WriteLine("Hello {0}", name);
            Console.ReadKey();
        }
    }
}
```

Using Statements

The first five lines consist of using statements. Using statements at the start of a program indicate which .Net libraries of code to use in the program. Of the ones listed here only "System" is necessary. The others are included in the Console template we used, but they actually aren't necessary. They are like advertisements. "Generic," "Linq," and "Threading" are libraries Microsoft recently added to .Net. They are placed here as sort of a notice that they are new and available to be used. "System.Text" does provide some additional tools for manipulating fonts and text that we could use, but our current program does not.

Namespaces

After the using statements we encounter a namespace. Namespaces have a dual function. On the one had they group things together that belong together. So all the code, and all the classes (we will talk about classes in a moment) that go together are grouped under this namespace. It also serves the reverse function—it separates things that do not belong together, and keeps them from being confused. By default, the Console template in Visual Studio makes the namespace the same as the name you gave the program when you first created it. You can change the namespace without problem if you wish.

For example, every Console application using the template in visual studio will have a "Program" class. Namespaces help keep Visual Studio and others, including yourself, from confusing them. You can see this using the "dot" notation. In C# and in most C based languages you use a dot to show belonging. For instance FirstCSharp.Program shows that this Program class belongs to the namespace "FirstCSharp."

We will use this kind of notation all the time. We will see it again inside the method static void Main, but that is getting ahead of ourselves.

For all its usefulness, a namespace is not required. We could have just started with class Program and left off the namespace. Everything would work just the same.

Curly Braces.

Right after the namepace, there is a curly brace {. Curly braces mark the beginning and ending of blocks. So the curly brace right after namespace is closed at the very end of the program. The curly brace that begins the class, is closed right above the one that closes the namespace. And the curly brace that begins the void Main is closed above the one that closes the class. These braces must be nested properly. Missing one, or placing one out of order will totally disrupt your program. Initially, nothing in programming is likely to give you as much trouble as curly braces. After a lot of practice though, they will become second nature.

At first, I would suggest you comment the braces to keep them straight. Comments are preceded by two slashes //. Here is the code with the end braces commented:

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

namespace FirstCSharp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Enter your Name");
            string name = Console.ReadLine();
            Console.WriteLine("Hello {0}", name);
            Console.ReadKey();
        }//end Main
    }//end class
}//end namespace
```

We will encounter other blocks set off by curly braces as we progress. If statements and loops are other blocks marked by curly braces.

Classes

The next line of code is the declaration of the class “Program”. A class is the definition of a type or an object. I know that is a bit vague at this time. C# is an object oriented language. Everything is divided into classes. You can’t have any code, other than using statements and namespaces, that is not contained in a class. Later we will talk more about classes, the built in ones and how to create our own.

Classes contain fields (variables), methods, and properties. We will talk about fields and properties later, but the next element in our program is a method, though a special one.

The method static void Main

The method static void Main is special for a couple of reasons. One, it is the starting point of the program. Every C# program has exactly one method called static void Main and it is always the starting point of the program. The function of Main is to call the first method or class that begins the execution of the program. Secondly the method is "static." That means that it's automatically loaded into memory when the program begins to run. Static methods and static classes are loaded into memory immediately when the program begins and stay in memory until the program ends.

There are several static classes that are available to us. Console, which we use in our program, is one of those. Because it is static, we don't have to initialize it. It is already there and available for use.

You may wonder, why not make every class static? In general it is not a good idea. Most classes and most methods just need to do their thing and then go away. Non static methods and classes are loaded into memory as they are needed and when they are done the memory is released for other classes and operations to use. Static classes stay in memory for the whole duration of the program. You should only use static for methods or classes that need to be available throughout the program execution, like Console.

Every method has parenthesis ( ) in which a programmer can place arguments to be passed into the method. In Main the parenthesis contain the parameter string[ ] args. This designates an array of strings with the variable name args. An array is a collection of related items of the same kind. We will deal with arrays later. If you start this program from a Dos prompt, you can pass in arguments by means of switches like:

```
Program –Steve –Conger
```

There would have to be code in the main to handle these values that you pass in. We are not going to be doing this.

Commands

Finally, in the Main method we get to the heart of program, the things it actually does. Here is where we write our commands. (Later we will make other methods to store our commands and reserve main for just calling the first method.)

A command is a statement telling the computer to do something. Our first command tells the computer to user the WriteLine method of the Console object to write a string onto the screen. It uses the dot notation to do this. As I mentioned above, Console is a static object that Microsoft built for .Net to handle input and output in console windows. The dot shows belonging. WriteLine is a method belonging to Console. It can write a string (any list of characters and numbers) to the screen.

```
Console.WriteLine("Enter your Name");
```

Within methods commands are executed in the order they are written. It is extremely important to get the order correct. Here we are doing things in this order:

1. Prompting the user for his or her name
2. Storing the name in a string variable called "name" (more in a minute)
3. Outputting the word Hello plus the name to the console

4. Waiting for a key stroke to end the program

If you were to do these in a different order, if, for example, you put the command to print the Hello and name to the console before you got the user's input. The program wouldn't do what you wanted. The name would never appear in the console.

The sequence of the commands also reveals a fundamental aspect of programming: Input, algorithm, output. In every program, you determine what needs to be brought into the program for it to work, in this case the users name, then you determine what needs to be done to that input to create the desired output—this is called the algorithm. In our program it is very simple, we combine the word "Hello" with their name to output it as "Hello name," where name is replaced by whatever they enter.

If you are ever stuck on where to start with a program think about what inputs does the program need? How are you going to get them? What do you need to do to the inputs to get the desired output?

Variables

When you are running a program, you often have need to store a value temporarily in memory. You could assign it to a memory address something like H33F1002GAAC21. But then you would have to remember that address to recall the value and there is always the danger that you would accidently assign two values to the same memory address. It is better to just give a name to thing you want to store and let the operating system determine where to store it.

Variables are basically names you give to remember values you are storing temporarily during the program. In our first program there is one variable. It is called "name" and it has a data type called "string."

When you declare a variable in C# you always give it a type and a name. The name can be anything as long as it doesn't contain spaces or special characters. The type lets C# know how much memory to reserve. Here are some possible variable declarations:

```
int number;

string city;

double price;
```

To assign a value to a variable you use the equal sign =.

```
price = 12.50;
```

(For this reason C# uses == to express equality as we will see later).

You can declare a variable and assign it a value in the same statement. This is how we did it in our program.

```
string name = Console.ReadLine();
```

Programming languages have "naming conventions." These are standards for how to form class names, method names, variable names etc. In C# the convention is to start variables with a lower case letter and then capitalize the beginnings of any additional words. For instance

```
double priceOfMeal;

string nameOfUser;
```

We will look at other naming conventions as we look at other elements.

Console Placeholder

There is one last element that needs explained. It is the {0} in the last Console.WriteLine(). This is a placeholder. The Microsoft programmers that made the Console object made it easier to display variable in the WriteLIne method. You can enter literal text and then where you want to insert the value of a variable you can use the braces and an index. After the quoted text, you have a comma and then the variable that will be inserted into the first placeholder. If you had a second placeholder you would put a 1 in it {1} and add another comma and anther variable.

Console.WriteLine("The price of the meal was {0} and the tax was {1}", price, tax);

The variable price would go into 0 and tax would go into 1. It doesn't know what the variables mean. It simple puts first into the first slot and second into the second slot. Indexes in C# always start with 0.

The final Console.Readkey(); is just to pause the program long enough to see the results. Once a key is pressed the program will end.