Aaron Schendel

Dr. Zelle

CS 230

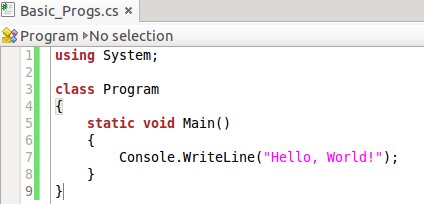
3 November 2011

Case Study – C#

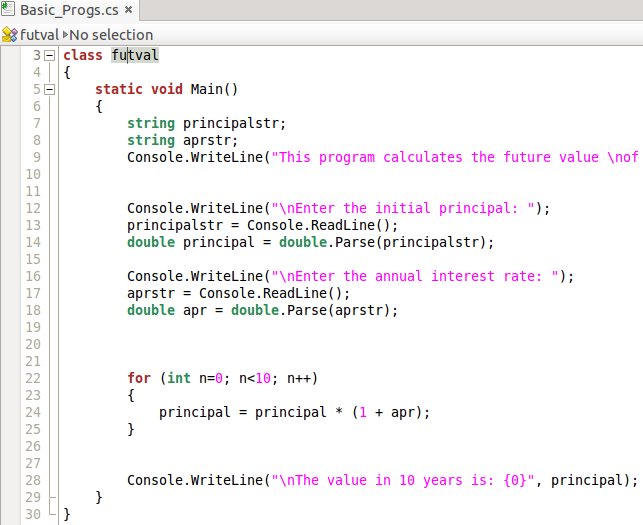
Basic Program Examples

If one wants to understand C# fully they must obviously understand not only the history of C# and how to obtain the environment, but also how to actually do some programming in this language. The best way to get an idea of what C# looks like is to take a look at some programming examples and break them down line by line. All of the following program examples will be shown in the MonoDevelop editor previously mentioned.

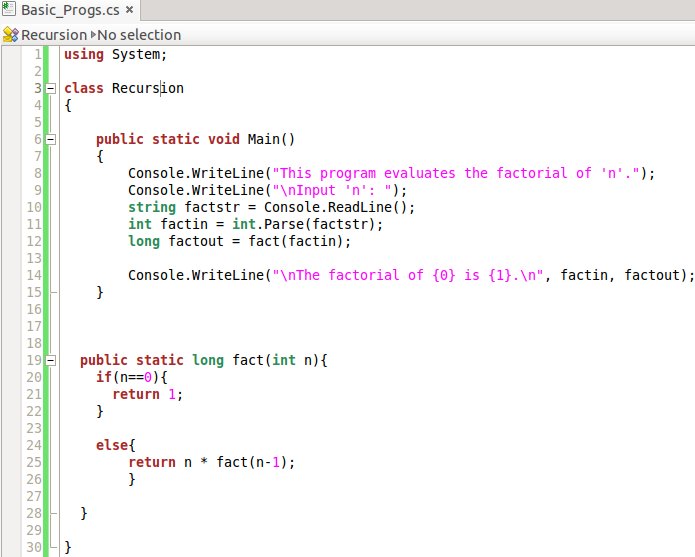
The first example is going to be extremely straightforward, a standard “Hello, World” function.



This is the same standard “Hello, World!” function that is ritually given as the first program example in most languages. In line 1 the word “using” is a keyword that simply states that we are going to be using functions from a defined library named whatever comes immediately after “using,” in this case it is the library “System.” This allows the programmer to not have to use prefix “System.” before every function from the “System” library. At the end of the line we have to include a semicolon to signal the end of a line, comparable to Java and various other languages. In line 3 we create a new class which we name whatever we would like, in this case “Program.” Line 4 is simply an opening brace to start the body of the class “Program.” Whether the brace immediately follows the class name or is on the next line is irrelevant. In line 5 we start out by declaring our method to be static which means that it is accessible without an instance of Program. We will always do this for our Main method because otherwise the program would require an instance, but any instance requires a program which would be a never-ending loop (Wikipedia). We then declare “void” before Main to say that it has no return value. In line 7 we arrive at the heart of this incredibly simple program. “Console.WriteLine” is saying that we are using the WriteLine method from the class Console. This is an example of where using “using” in line 1 saves time because without it we would have needed to type “System.Console.WriteLine” to get the same effect because Console is a class in System. Next we do just like is done in Python, Java, and various other languages: when we call a display method we have open-paren followed by a quotation followed by the string to be displayed followed by an ending quotation mark ended with a close-parenthesis and a semi-colon to end the line. We close both sets of brackets and now when this program is ran it will display the text “Hello, World!”

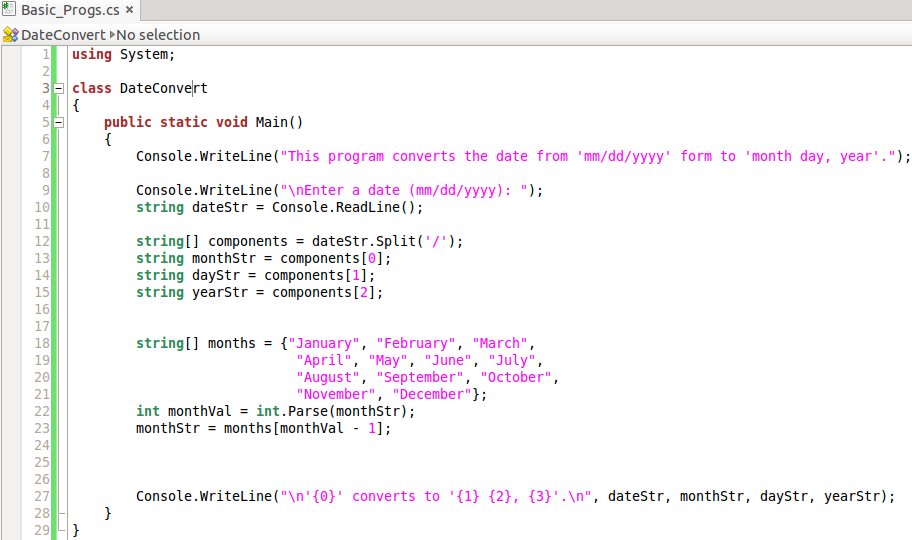
The next program we will look at is one that calculates the future value of an investment (Zelle 48-49).

This example clearly exemplifies some things that weren't in the previous program. Lines 1-6 are all the same as in the “Hello, World” program we just looked at but line 7 is something new. It is a variable type declaration followed by the variable declaration. This line is saying that we are initializing a variable name “principalstr” and setting it as type “string.” This is the same syntax that Java uses. Line 8 is the same thing but for a second variable and lines 9 (cut off due to picture size), 10, and 16 are just using the “WriteLine” method we used in the last program only this time it is used to display an introduction to our program and prompts for the variables. Similar to Java and Python if one puts a “\n” inside of a string it moves the rest of that string to the next line; this is being done in lines 9, 12, 16, and 28. In line 13 we are using a new method of Console called ReadLine() which prompts the user to enter a string, which we then stored as the previously initialized variable “principalstr”. Then in line 14 we evaluate that string as a double using the C# version of Python's “eval()”, “double.Parse()” and store that in the variable “principal”. The word “double” here can be replaced with “int” if the desired evaluation is an integer but for this program we are going to want doubles. This is a good example of how it is up to the programmer on whether or not to initialize all the variables at the beginning of the program (see line 7) or to simply initialize them as they are needed (see lines 14 and 18). Lines 16-18 are the same as 12-14. In line 22 we are introduced to the For Loop. The For Loop in C# is structured the exact same way as it is in Java where we use the keyword “for” followed by an initialization of n (n starts at 0), how high we want n to go (less than 10), and lastly how we want to increment n (adding 1 to n each time). The semi-colons in-between each part do indicate that they could each be on their own line but common programming convention puts them all next to each other when forming a For Loop. In line 24 we see that expressions are the same as we would see in Python or Java. Ending the meat of the program is line 28 where we display the results of our program by using WriteLine() but this time using more than one argument. When there is more than one argument needed to display we can use WriteLine() and put a string in and use “{0}, {1},...,{n}” to stand for the amount of arguments used after the quotation marks. When this program is ran it will display line 28 as “(new line) The value in 10 years is: (the value of principal).

Our next example that we will take a look at is a simple recursive factorial function.

When we look at this example we realize that it is different than the past two because this program implements the use of a helper method instead of putting everything in the Main. Lines 1-6 are the same as the previous two programs, we are just stating that we're using System, creating a class, and creating a Main method. In lines 8 and 9 we are just introducing the program and asking for them to input a value that, in line 10, we are storing as the variable “factstr” and then evaluating to an integer in line 11. It is in line 12 that we see our first method call in C#, however it is nothing too shocking since it uses the same syntax as Java. We can see in that same line that we are storing the return value of that method as the variable “factout” which is of type “long”. We are storing it as type “long” because it is equipped to handle much larger integers than that of type “int”. Now we get to the core of this program, the recursive factorial algorithm starting in line 19. We see in this method declaration that instead of using the “void” as a return value like in the previous examples, this time we need to return a value so we state that we are returning a value of type “long”. We see our first C# control structure in line 20, this “if” statement also looks the same as it's Java counterpart but something to take note of is that C# differs from C and C++ in the fact that in the condition statement for it's control structures it is necessary to have a statement that has a Boolean value. This is different than C and C++ because in both of those languages we can put in the conditional statement an expression that evaluates to 0 to equal false, or an expression that evaluates to any other number to equal false. In C# the conditional must evaluate to a Boolean value (Mayo). We also see in line 20 that we are using the double equals sign to check if “n” is equivalent to the value 0. Note that if we were comparing strings we would want to use “n.equals(0)” since the double equals will just be looking for numerical values. In line 21 we are just stating what we want to return if the “if” part of our control structure is true. Otherwise the “else” in lines 24-25 will run which returns the value of recursively running our “fact” method, decreasing “n” by one each time so that the recursion will end when “n” has a value of 0.

Our last example is a program that converts the form of a given date (Zelle 145).

In this program we deal with string manipulation and creating and accessing arrays. Lines 1-6 are the same as the previous examples. In lines 7 and 9 we are introducing our program and asking for the user to input a date in a very specific form. We ask for this form, the numbers separated by forward-slashes, so that in line 12 we can create a variable called “components” of type string-array that we are then able to store each individual part of the date in by separating the user input string at each forward-slash. Then in lines 13-15 we index the three strings in the array “components” and assign them to three separate variables of type string. We now have full access to the three separate parts of the input.

In line 18 we create a string array variable called “months” and fill it with the twelve months of the year so that we can index it in line 23. Lines 18-21 show that unless there is a semi-colon, the compiler will just treat everything as the same line which allows me to make the array easier to read by putting it on multiple lines. In line 22 we are evaluating the value of “monthStr” and storing it so that we can use it in line 23 to index “months” at the value the user entered. This brings up the point that once a variable is initialized as a type you can change the variable's value but you cannot change its type. Also, the reason we are subtracting one is because the months are numbers 1-12 whereas the array is numbered 0-11. In line 27 we see an excellent example of how handy output formatting can be.

Bibliography

Mayo, Joe. "C# Tutorial." *C# Station*. 2011. Web. 02 Nov. 2011. <http://www.csharp-station.com/Tutorial/CSharp>.

Mono Development Team. "What Is Mono." *Mono*. Xamarin. Web. 20 Oct. 2011. <<http://www.mono-project.com/What_is_Mono>>.

Mono Development Team. "General Questions." *MonoDevelop*. Xamarin. Web. 20 Oct. 2011. <http://monodevelop.com/>.

Sebesta, Robert. *Concepts of Programming Languages*. Ninth. Pearson Education, Inc., 2009. 103-4. Print.

Unknown. "Variables." *.Net Tutorials*. 2011. Web. 02 Nov. 2011. <http://csharp.net-tutorials.com/>.

Zelle, John. *Python Programming: An Introduction to Computer Science*. 2nd. Wilsonville: Franklin, Beedle & Associates Inc., 2010. 48-49, 145. Print.