Welcome to Learn C# – Week 2

Previously on Learn C#, we found out how to compile and write messages on the screen. This time around we are going to learn several different aspects of programming that will allow us to make our programs a little bit more interactive. Now because this is a gaming website and am sure most of you reading this have at some point played a RPG or a Pokémon game I thought it would be practical to make some demos that you could see work in those games. This week we are going to make a program that prompts the user about their information like Professor Oak does in the beginning of a Pokémon game.

Alright so recall that we know how to print messages on the screen using **Console.WriteLine(“”);** and can keep that message on screen using **Console.ReadLine(“”);.** Read line’s functions isn’t to keep a message on screen it’s to take in data from a user. But how do we store that data? We will have to use **variables**. I know that word probably scares a couple of you. You’re used to hearing the word variable in a math class usually related to an equation like 5 + X = 10. That’s a simple example and if you don’t know the answer to that it’s 5. X is a missing number that you usually have to discover. In our case we won’t be trying to discover what X is, we will place a value in X. Let’s test this out in code.

Let’s open up Visual Studio and create a new project by going to File in the upper left corner, then hitting New, then Project. Look at the Week 1 link if you forgot how to do it. Alright once you set up the file type the following in the code in between the static void Main curly braces.

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

{
int x = 0;

}
```

The code int x = 0; Did several things at once.

1. It created a variable named x.

2. It created a variable of type integer.
3. It assigned a value to the x of the number 0.

Let me elaborate on each section. First, you can call a variable anything you want. In programming it's generally common to name your variables something descriptive. Naming a variable x, y, z or something without a meaning will most likely make a person trying to read your code confused on what you are trying to do and later on when you have hundreds of lines of code you yourself will most likely forget what a variable with just a x does. So change the code above and replace the variable name X to number. It's coding practice that variable names start with lowercase letters. Your compiler won't throw complaints if you capitalize variable names but it is generally accepted by all programmers to lowercase your variables.

Change x to number like shown below. Notice how variable names can be anything and you won't get any issues. It's just a name for a place in memory where we will store data.

```
int number = 0;
```

Second, is we put the word **int** in front of our code. This tells the compiler that the data we are putting in the variable is going to be an integer. According to Microsoft, the range of numbers we can express with **int** is -2,147,483,648 to 2,147,483,647. Meaning if we try to pass the positive number 2,147,483,647 we will have a overflow. If we go any lower than the shown negative number we will have an underflow. Now you may be asking "What the heck does that even mean?" well a gaming example of an underflow comes from an old video game story. I've forgotten the name of the game but this story is about a player who was fighting a Boss maybe in a Final Fantasy game. The player was able to cast heal on the opponent until the number of health points HP, which is likely kept track of with a variable named healthPoints in the game overflowed. What happened when the enemy Boss health overflowed? The number went from the positive limit in our case 2,147,483,647 all the way to -2,147,483,648. The player healed the boss so much that it surpassed the limit of the **int** type and went and looped to the negative side which meant the boss was dead since his HP is below 0. Stories like these are not uncommon in programming. You can try to bullet proof your code as much as you want but you can never really predict what a user/gamer will try to attempt with your code and like I said in the first lesson, you're going to need to try to think of every conceivable step.

int is just one of the many types that C# offers. You can try changing **int**to **long** which offers a bigger range of numbers to use. You can try **float** which allows you to add decimals and regular integers but the range of numbers you can represent goes down even lower than **int**. Below is a list of some of the types we will use in our lessons.

- **float** : Decimal numbers like 2.5
- **double** : Even More Decimal Numbers
- **int** : Integers. We won't go past the limit of int so we will stick with this type instead of **long**.
- **char** : We can use char to store characters like 'a', 'b', 'c' and many more. You can't store a integer like the number 2 using char.

- **string** : With the String type we can store words like “Hello” in variables.
- **bool** : bool stands for Boolean which is the type that can store values 0 and 1 which represent false and true.

Alright so now that we know all of that the last part of the code is the **assignment operator** =. The equals sign is used in programming to assign a value to variables. The = sign does not mean “is equal” there is another representation of that in programming. The assignment operator just gives the value of whatever is on the right side to the left. So because we have a 0 on the right side it gives that value to the left hand side of the assignment operator = which is our variable x.

Here are some extra operators that we will use

+ :Unlike the assignment operator these operators work like they do in real life. This one adds.

* :This operator is used for multiplication

/ :This operator is used for division

Now that we know how it all works lets test it out.

```
namespace Week2
{
    class Program
    {
        static void Main(string[] args)
        {
            //Set the variable number of type int to 0
            int number = 0;

            //assign the value of 5 + 5 to the variable number
            number = 5 + 5;

            //display the new value of the
            Console.WriteLine(number);

            //Pause the screen to display number
            Console.ReadLine();

        }
    }
}
```

Note that every line of code ends with a semi colon. If your code doesn’t end with a semicolon the entire line will be seen as an error. Also not that I added comments by putting //before writing regular text. Putting to dashes tells the compiler to ignore anything after the // on that line.

Alright debug and compile your program by pressing the Green Arrow Start button. If you typed everything correctly you should see the number 10 appear. If it doesn’t it means you

mistyped something. Remember that you should only change the stuff in the static void Main. My program has Week2 on top because that's what I decided to name my project, that shouldn't affect your program.

Now some of the things you should keep in mind is that you can change a variable's content at anytime. We had it initially set to 0 but could have started the program by not giving it a value at all. We could have just typed

int number;

and that would have worked just fine. The only problem with that is we have no idea what's inside that variable and it could very well be some junk value in memory so that it doesn't cause any troubles I personally like to always set a value at the start. Second thing to note is that we don't have to put the type of a variable after we **initialized(gave it a value)** it. We didn't put int every where number is at only when we first created the variable.

Last thing to notice is that Console.WriteLine(); doesn't need quotations inside the parenthesis. Remember before in our Hello World program we wrote Console.WriteLine("Hello World"); we wrote "Hello World" with parenthesis but that's only necessary for **strings**. Because we used a variable we didn't need to put the variable number in quotes. If we wanted to display a number without using a variable we would still have to use quotes. Console.WriteLine("5");

That's all cool and all but your probably wondering? What does this have to do with Pokémon and the Professor Oak program we were going to do? Well now that you know the basic building blocks we can continue and do just that. Below the current code you wrote or even just replacing everything write the following.

```
string name = "Red";
string gender = "blank";
Console.WriteLine("Welcome to Pokemon");
Console.WriteLine("Enter your name.");
name = Console.ReadLine();
Console.WriteLine("Are you a boy or a girl?");
gender = Console.ReadLine();
if (gender == "boy") {
    Console.Write("Your name is ");
    Console.WriteLine(name);
    Console.Write("You are a ");
    Console.WriteLine("boy");
}

if (gender == "girl")
{
    Console.Write("Your name is ");
    Console.WriteLine(name);
    Console.Write("You are a ");
    Console.WriteLine("girl");
}
Console.ReadLine();
```

Line by line, the first line creates a variable of type string initialized to a default name Red just like the default name in the Pokémon games was either Red or Blue. Professor Oak asks for the gender of the Pokémon trainer so we created another variable for gender. Next we display the main title message of our program “Welcome to Pokémon.” We then tell the user to enter his name and then use Console.ReadLine(); to get the value the user enters all the way until he presses the enter key. We take that name the user types and using the assignment operator put it in the name variable. We next ask if the trainer is a boy or a girl. We take the input which has to be either boy or girl written exactly as is without capital letters. Next I introduce something we will go more in depth next week which is the **conditional If branch.** the If branch checks whether or not what the user enters is true. If it’s true the code inside of its curly braces is executed, if it isn’t it skips over that code. In our case it checks whether or not the user entered boy or girl and skips one of them depending on what the user entered. If you wrote everything correctly it should work fine. Remember this, Visual Studio is more than likely never the one that will have issues. It will be you that makes the mistakes and it could very well be a tiny mistake like a forgotten semi colon that stops the entire program from working.

What you Learned

- Data Types: int, string, double, and char are used to tell what type of data you are working with
- Variables: They store the data we will be manipulating.
- Initialization: We can immediately set the value of variables.
- Operators: We can use the assignment operator to give variables values or use operators like +,-, * and more to manipulate data.
- Conditional If branch: We will go over this and more conditional branches next week but know that these are some of the most powerful sets of tools you will ever get to use in any language.

Homework

Alright so you know a lot now. So much so that you can probably make your own program if I gave you a set of instructions. Try doing the following. Try making a calculator program.

1. First prompt the user and display the message “Welcome to the Basic Calculator”
2. Ask the user to enter two numbers.
3. Use Console.ReadLine();(<-Ignore This) The actual command to read integers is **Console.Int32.Parse(Console);** Some people may have had issues just using the standard ReadLine(); and take in two numbers a user enters and puts them inside two separate variables of type int.
4. Then add those two variables together and use the assignment operator to put that result inside a third variable.
5. Lastly display that third int variable using Console.WriteLine();

You learned how to do all of this so try on your own. If you want you can look for help online if your stuck or even ask a question below. I’ll post a possible solution in next weeks lecture but your code can look completely different than mine. As long as it takes into two values adds them and then displays those values on screen you know you passed the homework. Also you won’t have to use conditional If branches in this homework so don’t even try them.