Binary Operators

This lesson takes a look at the binary operators in C# in detail using coding examples

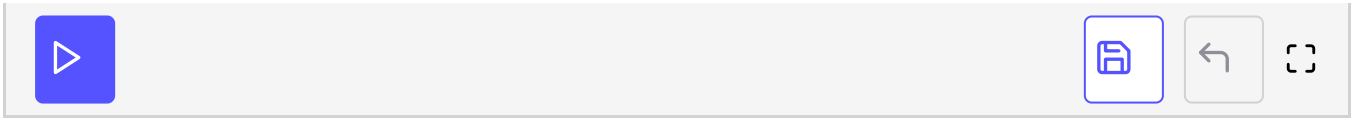
WE'LL COVER THE FOLLOWING

In the previous lesson you studied **unary** *operators*, in addition to those C# has **binary** *operators* as well that form expressions of *two* variables. The example below shows how to use the binary operators.

Binary Operators Example

```
using System;

class Binary
{
    public static void Main()
    {
        int x, y;
        float result;

        x = 9;
        y = 4;

        result = x+y; //addition
        Console.WriteLine("x+y: {0}", result);

        result = x-y; //subtraction
        Console.WriteLine("x-y: {0}", result);

        result = x*y; //multiplication
```

```
        Console.WriteLine("x*y: {0}", result);

        result = x/y; //division

        Console.WriteLine("x/y: {0}", result);

        result = (float)x/(float)y; //casting int to float and then performing division
        Console.WriteLine("x/y: {0}", result);

        result = x%y; //modulus
        Console.WriteLine("x%y: {0}", result);

        result += x; //assignment with operation
        Console.WriteLine("result+=x: {0}", result);
    }
}
```



Binary Operators

Code Explanation

The example above shows several examples of **binary** operators. As you might expect, the results of

- *addition* (`+`)
- *subtraction* (`-`)
- *multiplication* (`*`)
- *division* (`/`)

produce the expected mathematical results.

- In the code the `result` variable is a **floating** point type.
- We explicitly cast the **integer** variables `x` and `y` to calculate a floating point value in line **25**.
- There is also an example of the *remainder*(`%`) operator.
    - It performs a **division** operation on two values and returns the *remainder*.
- The last statement in line **31** shows another form of the assignment with operation (`+=`) *operator*.

> **Note:** Any time you use the **assignment with operation operator**, it is

> **Note:** Any time you use the **assignment with operation operator**, it is the same as applying the **binary** operator to both the **left**-hand and **right**-hand sides of the operator and putting the results into the **left**-hand side.

The example could have been written as

```
result = result + x;
```

and returned the same value.

Computing Area of Circle

Now let's take a look at another example. In the code below we will compute the *area* of a **circle**.

```
using System;

class Area
{
  static void Main()
  {
    double pi = 3.14159;
    double radius = 2.5;
    double area = pi * radius * radius; //computing area of circle
    Console.WriteLine("The area of a circle with radius " + radius + " is " + area);
  }
}
```

Area of a Circle

Code Explanation

In the example above

- In line 7 we've assigned a value for the constant `pi`.
    - `pi` is required for finding the **area** of a circle.
- In line **8** we've also assigned a constant value for the **radius** of the circle.

Line **9** is where the computer gets to work number crunching.

- The **area** of a circle is given as `A = pi * r * r`.
- The *result* of the calculation will be stored in a variable named `area`.
- In line **10** we output the *result* of the calculations to the console. As a bonus, we've also output the *radius* on the circle.

User Input Example

A programmer can request the user to input a value in the console.

A line must be stored in a **C#** type called a **string**.

- A string contains a list of `chars`.
- This string must then be converted to the type that was requested by the program.

The best way to learn is by example and by doing, so here is an example of reading a **string** and then converting that *string* to another *data* type.

```
string var = Console.ReadLine();    // stores the users input into a string

double radius = double.Parse(var);  // converts the string into a double using the Parse meth

double radius = double.Parse(Console.ReadLine()); // same as the previous method

int value = int.Parse(Console.ReadLine());  // converts to an integer instead of a double

float value = float.Parse(Console.ReadLine());        // converts to a float
```

Each **C#** data type has a method to parse a **string**, and convert that string into the corresponding *data* type.

The problem, however, is that the user might make a mistake and enter a string that cannot be parsed.

An example would be if the program requested an **integer** value, but the user entered a **floating** point value. The program will then throw an error that the *Input* string was not in a correct format.

For now, we will assume that the user will always enter the correct values.

Example

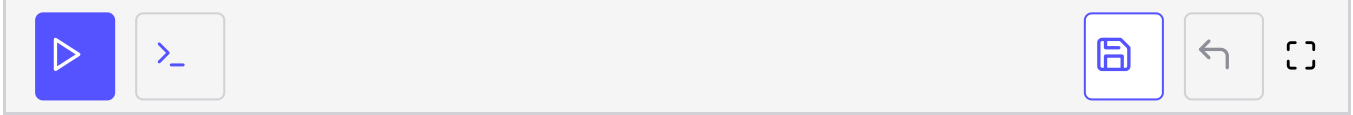
In the example above we calculated the **area** for a given **fixed** *radius*. However, now in the example below, we will use the **user input** to pass in any random **radius** and our program will calculate the **area** for that *radius*.

Due to some restrictions of our coding platform, you will need to follow the following steps before running the code below:

- Click on the `>_STDIN` button located beside the `RUN` button in the code widget below.
- An input bar will appear.
- Enter the value of **radius** for which you want to calculate the area.
- Finally click the `RUN` button.

> **IMPORTANT NOTE:** If you run the example below without passing the radius through the command line as explained above you will get an error.

```
using System;

class Program
{
    static void Main()
    {
            double pi = 3.14159;
            Console.Write("Please enter a value for the radius: ");
            double radius = double.Parse(Console.ReadLine());
            double area = pi * radius * radius;
            Console.WriteLine("The area of a circle with radius " + radius + " is " + area);
    }
}
```



Passing Radius Through Command Line

Code Explanation

- First we prompt the user for a value
- In line **9** the program reads the *value* passed by you through the console using the `Console.ReadLine()` function and stores it in the variable

named `radius`

    - The *value* passed by you is in the form of a **string**
    - The `double.Parse()` function in line **9** converts the **string** to type `double`

- In line **10** this `radius` is used to *compute* the `area` of the *circle*
- In line **11** this `area` is then displayed to the console along with the `radius` passed

This marks the end of our chapter on the types of *operators* in C#. Interesting so far? Well then read on to the next chapter to learn more exciting stuff.