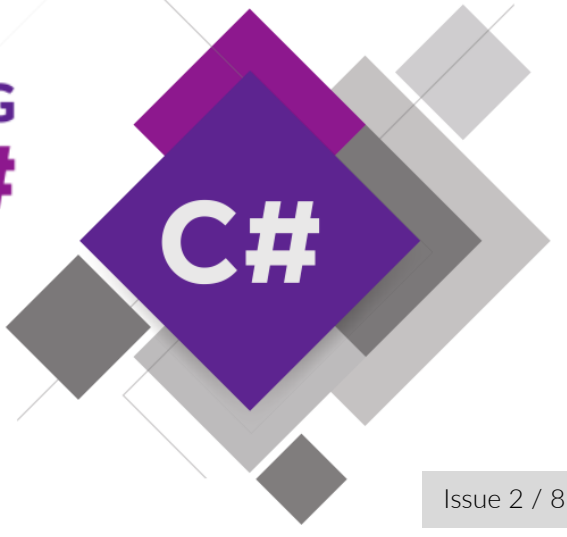
PROGRAMMING BASICS WITH C#



Issue 2 / 8

SIMPLE CALCULATIONS

Learn about performing simple calculations in C#, using the system console, defining and using variables, expressions and calculations, using data types, arithmetic operators, reading and printing text and numbers on the console.

Get an idea how to build simple Desktop GUI app "Currency Converter" in C# using Visual Studio.

Dr. Svetlin Nakov

Software University – https://softuni.org

Simple Calculations

In this mini book we are going to get familiar with the following concepts and programming techniques:

- What is the **system console**?
- How to **read numbers** from the system console?
- How to work with **data types and variables**, which are necessary to process numbers and the operations between them?
- How to **print** output (a number) on the console?
- How to do simple **arithmetic operations**: add, subtract, multiply, divide, string concatenation?

Video: Overview

Watch a video about what shall we learn in this mini book here: https://youtu.be/NXbFJw_NstA.

Introduction to Simple Calculations by Examples

Computer programs can **enter data** from the **console**, perform **calculations** and **print the results** on the console. This is a simple example of C# program that **converts** from **foots** to **meters**:

```
Console.Write("Foots = ");
var foots = double.Parse(Console.ReadLine());
var meters = foots * 0.3048;
Console.Write("Meters = ");
Console.WriteLine(meters);
```

Run the above code example: https://repl.it/@nakov/foots-to-meters-csharp.

The above program **enters a number** and **converts** its value from **foots** to **meters**. This is a **sample output** from the above code, when the user enters `5` as input:

```
Foots = 5
Meters = 1.524
```

In C# we can **read a text line** from the console using `Console.ReadLine()` and we can convert the text to a floating-point number using `double.Parse(text)`. We can **print text and numbers** using the **$ text formatting** syntax as follows:

```
var radius = 1.25;
Console.WriteLine($"Circle radius = {radius}");
Console.WriteLine($"Circle area = {Math.PI * radius * radius}");
```

Run the above code example: https://repl.it/@nakov/circle-area-csharp.

The **$ syntax** replaces all expressions in curly brackets with their values. The output from the above code is:

```
Circle radius = 1.25
Circle area = 4.90873852123405
```

Let's explain in greater detail how to use the **console**, how to **enter numbers** and text and how to perform simple **calculations** and **format and print text** and expressions on the console in C#.

The System Console

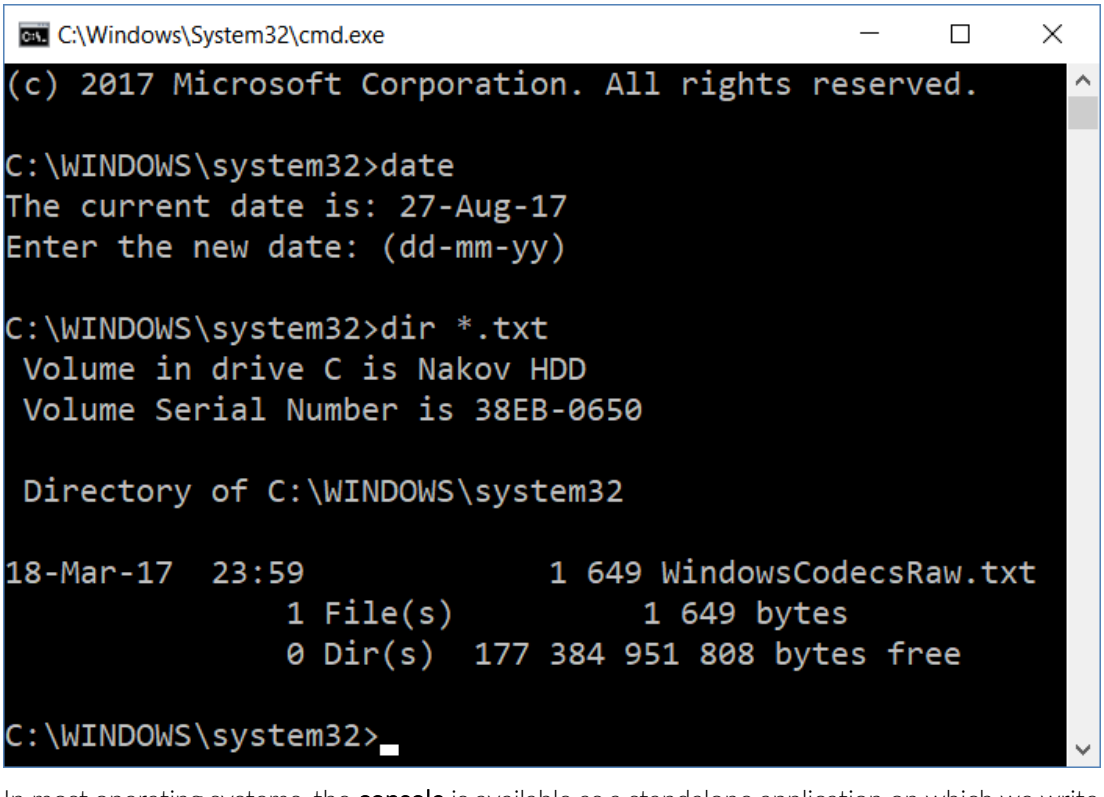
Simply called "**console**", the "system console", the "system **terminal**", also the "computer console", represents the tool by which we **give the computer commands** in a text format and **get the results** from their execution again as a text.

Video: The System Console

Watch a video lesson about the system console here: https://youtu.be/ehHnNu6M55M.

The System Console Explained

Generally, the **system console** represents a text terminal, which means that it accepts and visualizes just **text** without any graphical elements like buttons, menus, etc. It usually looks like a black colored window like this one:



In most operating systems, the **console** is available as a standalone application on which we write console commands. It is called a **Command Prompt** in Windows, and a **Terminal** in Linux and Mac. The console runs console applications. They read text from the command line and print text on

the console. In this book we are going to learn programming mostly through creating **console applications**.

In the next examples we will **read data** (like integers, floating-point numbers and strings) from the console and will **print data** on the console (text and numbers).

Reading Integers from the Console

In order to read an **integer** (not a float) **number** from the console, we have to **declare a variable**, declare the **number type** and use the standard command for **reading a text line** from the system console `Console.ReadLine()` and after that **convert the text line into an integer number** using `int.Parse(text)`:

```
var num = int.Parse(Console.ReadLine());
```

The above line of C# code **reads an integer** from the first line on the console.

Video: Reading Data from the Console

Watch a video lesson about reading from the system console here: https://youtu.be/WPlQ5HYBGJQ.

Video: Reading Integers from the Console

Watch a video lesson about reading integer numbers from the system console here: https://youtu.be/3TC2F-ffw34.

Example: Calculating a Square Area

For example, let us look at the following program, which **reads an integer from the console**, multiplies it by itself (**squares** it) and **prints the result** from the multiplication.

Video: Calculating a Square Area

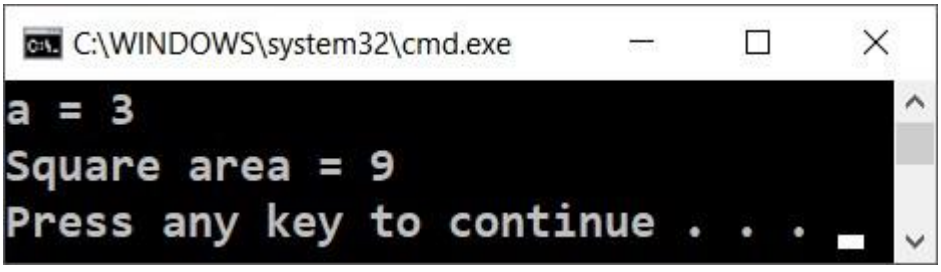
Watch a video lesson about calculating square area: https://youtu.be/gdYTotTFVgA.

Code: Calculating a Square Area

This code demonstrates how we can calculate the **square area** by the given length of the side:

```
Console.Write("a = ");
var a = int.Parse(Console.ReadLine());
var area = a * a;
Console.Write("Square area = ");
Console.WriteLine(area);
```

Here is how the program would work when we have a square with a side length equal to 3:



Try to write a wrong number, for example "hello". You will get an error message during runtime (exception). This is normal. Later on, we will find out how we can catch these kinds of errors and make the user enter a number again.

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#0.

How Does the Example Work?

The first line `Console.Write("a = ");` prints an informative message, which invites the user to enter the side of the square `a`. After the output is printed, the cursor stays on the same line. Staying on the same line is more convenient for the user, visually. We use `Console.Write(…)`, and not `Console.WriteLine(…)` and this way the cursor stays on the same line.

The next line `var a = int.Parse(Console.ReadLine());` reads an integer from the console. Actually, it first reads a text (string) using `Console.ReadLine()` and after that it gets converted to an integer (it is parsed) using `int.Parse(…)`. The result is kept in a variable with name `a`.

The next command `var area = a * a;` keeps in a new variable `area` the result of the multiplication of `a` by `a`.

The next command `Console.Write("Square area = ");` prints the given text without going to the next line. Again, use`Console.Write(…)`, and not `Console.WriteLine(…)`, and this way the cursor stays on the same line in order to print the calculated area of the square afterwards.

The last command `Console.WriteLine(area);` prints the calculated value of the variable `area`.

Data Types and Variables

In programming, each variable stores a certain **value** of a particular **type**. For example, data types can be: **number**, **letter**, **text** (string), **date**, **color**, **image**, **list** and others. Here are some examples of data types:

- **integer**: 1, 2, 3, 4, 5, 20, …
- **float**: 0.5, 3.14, -1.5, …
- **character** (symbol): 'a', 'b', 'c', '@', 'X', …
- **text** (string): "Hello", "Hi", "Beer", …
- **day of week**: Monday, Tuesday, …, Sunday
- **date and time**: 14-June-1980 6:30:00, 25-Dec-2017 23:17:22

Video: Data Types and Variables

Watch a video lesson about declaring variables: https://youtu.be/p4tedmW8dyw.

Examples: Data Types and Variables

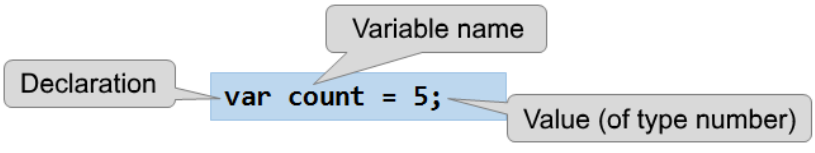
In C# we can use **data types** to define **variables** as follows:

```
int a = 5;
string str = "Some text";
char letter = 'A';
float f = 4.2;
```

In C#, once a **variable** is defined, it can **change its value** many times, but it **cannot change its data type** later. Variables may hold only data of their type.

Declaring and Using Variables

We know that computers are machines that process data. All **data** is stored inside the computer memory (RAM) in **variables**. The variables are named areas in the memory, which keep a certain data type, for example a number or a text. Each of the **variables** in C# has a **name**, a **type** and a **value**. Here is how we would **declare a variable** and **assign it** with a **value** at the same time:



Video: Declaring and using Variables

Watch a video lesson about declaring variables: https://youtu.be/g-dG5GobHg0.

Examples: Declaring and using Variables

Example of **declaring a variable**:

```
var count = 5;
```

After being processed, data is again stored in variables (in some place in the memory saved for our program):

```
count = count + 1;
```

After the above code the variable `count` changes it value and increases by `1`.

Reading Floating Point Numbers from the Console

To read a **floating-point number** (fractional number, non-integer) from the console use the following command:

```
var num = double.Parse(Console.ReadLine());
```

The above C# code first reads a **text line** from the console, then converts (parses) it to a **floating-point number**.

Video: Reading Floating-Point Numbers

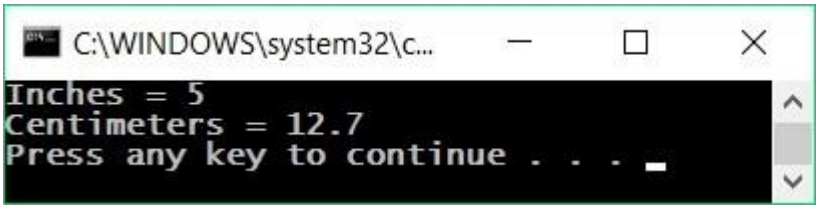
Watch the following video lesson about how to read floating-point numbers from the console: https://youtu.be/H2waLeIW70A.

Example: Converting Inches into Centimeters

Let's write a program that reads a floating-point number in inches and converts it to centimeters:

```
Console.Write("Inches = ");
var inches = double.Parse(Console.ReadLine());
var centimeters = inches * 2.54;
Console.Write("Centimeters = ");
Console.WriteLine(centimeters);
```

Let's start the program and make sure that when a value in inches is entered, we obtain a correct output in centimeters:



Note that if you enter and invalid number, e.g. "*asfd*", the program will crash with an error message (exception). We will learn how to handle exceptions later.

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#1.

Reading a Text from the Console

To read a **text** (string) from the console, again, we have to **declare a new variable** and use the standard **command for reading a text from the console**:

```
var str = Console.ReadLine();
```

By default, the `Console.ReadLine(…)` method returns a **text result** – a text line, read from the console.

- After you read a text from the console, additionally, you can **parse the text** to an integer by `int.Parse(…)` or a floating-point number by `double.Parse(…)`.
- If parsing to a number is not done, **each number** will simply be **text**, and we **cannot do** arithmetic operations with it.

Video: Reading Text from the Console

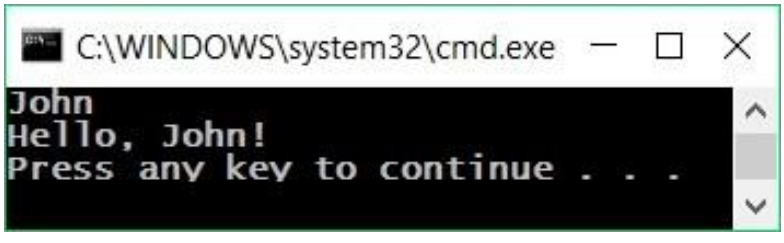
Watch a video lesson about how to read text from the console: https://youtu.be/0tzvEdWxZ1k.

Example: Greeting by Name

Let's write a program that asks the user for their **name** and salutes them with the text "**Hello, *<name>*!**".

```
var name = Console.ReadLine();
Console.WriteLine("Hello, {0}!", name);
```

In this case the `{0}` expression is replaced with the **first** passed argument, which holds the variable `name`. If we enter "*John*", the output will be as follows:



Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#2.

Printing and Formatting Text and Numbers

In C#, when printing a text, numbers and other data on the console, **we can join them** by using templates `{0}`, `{1}`, `{2}` etc. In programming, these templates are called **placeholders**. This is a simple example:

```
Console.WriteLine("{0} + {1} = {2}", 3, 5, 3+5);
```

The placeholders `{0}`, `{1}` and `{2}` are replaced by the expressions, given after the text. The result from the above code is:

```
3 + 5 = 8
```

Video: Printing Text and Numbers

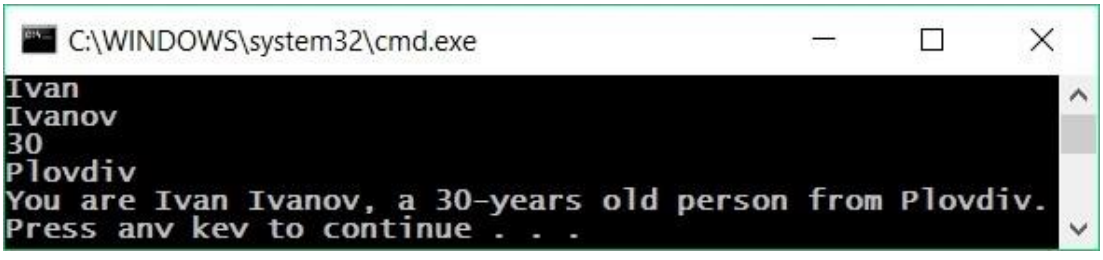
Watch a video lesson about how to print text and numbers together on the console: https://youtu.be/tSTwwaQpy9g.

Example: Printing Text and Numbers

```
var firstName = Console.ReadLine();
var lastName = Console.ReadLine();
var age = int.Parse(Console.ReadLine());
```

```
var town = Console.ReadLine();
Console.WriteLine("You are {0} {1}, a {2}-years old person from {3}.",
  firstName, lastName, age, town);
```

This is the **result** we are going to obtain after the execution of this example:



Notice how every variable should be passed in the **order, in which we want it to be printed**. Practically, the template (**placeholder**) **accepts variables of any type**.

It is possible for a template to be used **multiple times** and it is not necessary for the templates to be numbered sequentially. Here is an **example**:

```
Console.WriteLine("{1} + {1} = {0}", 1+1, 1);
```

The result is:

```
1 + 1 = 2
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#3.

Using the Dollar String Interpolation

We can format text in C# using also the following **$ syntax**. It provides simplifies text formatting:

```
var a = 4.5;
Console.WriteLine($"Square size = {a}");
Console.WriteLine($"Square area = {a * a}");
```

The output from the above code is as follows:

```
Square size = 4.5
Square area = 20.25
```

The `$` prefix before a string in C# enables the so called "**string interpolation**": replacing all expressions, staying in curly brackets `{ }` in the text with their values.

Using the **dollar string interpolation syntax**, the last example can be rewritten like this:

```
var firstName = Console.ReadLine();
var lastName = Console.ReadLine();
var age = int.Parse(Console.ReadLine());
var town = Console.ReadLine();
```

```
Console.WriteLine($"You are {firstName} {lastName}, a {age}-years old
person from {town}.");
```

Play with the above code and test it in the SoftUni online judge system: https://judge.softuni.org/Contests/Practice/Index/504#3.

Arithmetic Operations

Let's examine the basic **arithmetic operations** in programming. We can add, subtract, multiply and divide numbers using the operators `+`, `-`, `*` and `/`.

Video: Arithmetic Operators

Watch a video lesson about the arithmetic operators: https://youtu.be/XOtEuEUbA4M.

Summing up Numbers: Operator +

We can **sum** up numbers using the `+` operator:

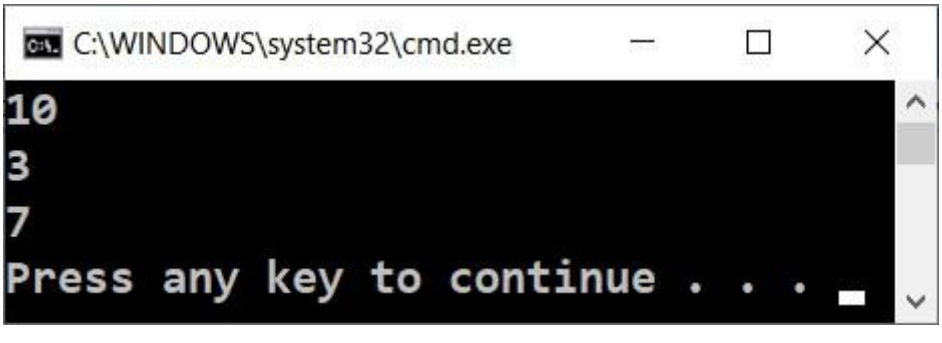
```
var a = 5;
var b = 7;
var sum = a + b; // the result is 12
```

Subtracting Numbers: Operator -

Subtracting numbers is done using the `-` operator:

```
var a = int.Parse(Console.ReadLine());
var b = int.Parse(Console.ReadLine());
var result = a - b;
Console.WriteLine(result);
```

Here is the result of the execution of this program (with numbers 10 and 3):



Multiplying Numbers: Operator *

For **multiplication** of numbers we use the `*` operator:

```
var a = 5;
var b = 7;
```

```
var product = a * b; // 35
```

Dividing Numbers: Operator /

Dividing numbers is done using the `/` operator. It works differently with integers and floating-point numbers.

- When we divide two integers, an integer division is applied, and the obtained output is without its fractional part. Example: 11 / 3 = 3.
- When we divide two numbers and at least one of them is a float number, a floating division is applied, and the obtained result is a float number, just like in math. Example 11 / 4.0 = 2.75. When it cannot be done with exact precision, the result is being rounded, for example 11.0 / 3 = 3.66666666666667.
- The integer division by 0 causes an exception during runtime (runtime exception).
- Float numbers divided by 0 do not cause an exception and the result is +/- infinity or a special value NaN. Example 5 / 0.0 = ∞.

Here are a few examples with the division operator:

```
var a = 25;
var i = a / 4;      // we are applying an integer division:
                    // the result of this operation will be 6 – the
fractional part will be cut,
                    // because we are dividing integers
var f = a / 4.0;    // 6.25 – floating division. We have set the
number 4 to be interpreted
                    // as a float by adding a decimal separator
followed by zero
var error = a / 0;  // Error: Integer divided by zero
```

Dividing Integers

Let's examine a few examples for integer division (remember that when we divide integers in C# the result is an integer):

```
var a = 25;
Console.WriteLine(a / 4);  // Integer result: 6
Console.WriteLine(a / 0);  // Error: divide by 0
```

Dividing Floating-Point Numbers

Let's look at a few examples for floating division. When we divide floating point numbers, the result is always a float number and the division never fails, and works correctly with the special values +∞ and -∞:

```
var a = 15;
Console.WriteLine(a / 2.0);   // Float result: 7.5
Console.WriteLine(a / 0.0);   // Result: Infinity
Console.WriteLine(-a / 0.0);  // Result: -Infinity
Console.WriteLine(0.0 / 0.0); // Result: NaN (Not a Number), e.g. the result from
```

```
                                    // the operation is not a valid numeric value
```

When printing the values ∞ and -∞, the console output may be ?, because the console in Windows does not work correctly with Unicode and breaks most of the non-standard symbols, letters and special characters. The example above would most probably give the following result:

```
7.5
?
-?
NaN
```

Concatenating Text and Numbers

Besides for summing up numbers, the operator + is also used for **joining pieces of text** (concatenation of two strings one after another). In programming, joining two pieces of text is called "**concatenation**". Here is how we can concatenate a text with a number by the + operator:

```
var firstName = "Maria";
var lastName = "Ivanova";
var age = 19;
var str = firstName + " " + lastName + " @ " + age;
Console.WriteLine(str);  // Maria Ivanova @ 19
```

Video: Concatenating Text and Numbers

Watch a video lesson about concatenating text and numbers: https://youtu.be/vPI-V2NG_CU.

Examples: Concatenating Text and Numbers

Here is another **example** of concatenating text and numbers:

```
var a = 1.5;
var b = 2.5;
var sum = "The sum is: " + a + b;
Console.WriteLine(sum);  // The sum is: 1.52.5
```

Did you notice **something strange**? Maybe you expected the numbers a and b to be summed? Actually, the concatenation works from right to left and the result above is absolutely correct. If we want to sum the numbers, we have to use **brackets**, in order to change the order of execution of the operations:

```
var a = 1.5;
var b = 2.5;
var sum = "The sum is: " + (a + b);
Console.WriteLine(sum);  // The sum is: 4
```

Numerical Expressions

In programming, we can calculate **numerical expressions**, for example:

```
var expr = (3 + 5) * (4 - 2);
```

The standard rule for priorities of arithmetic operations is applied: **multiplying and dividing are always done before adding and subtracting**. In case of an **expression in brackets, it is calculated first**, but we already know all of that from school math.

Video: Numerical Expressions

Watch a video lesson about numerical expressions: https://youtu.be/6MPxlOCsPdw.

Example: Calculating Trapezoid Area

Let's write a program that inputs the lengths of the two bases of a trapezoid and its height (one floating point number per line) and calculates the **area of the trapezoid** by the standard math formula:

```
var b1 = double.Parse(Console.ReadLine());
var b2 = double.Parse(Console.ReadLine());
var h = double.Parse(Console.ReadLine());
var area = (b1 + b2) * h / 2.0;
Console.WriteLine("Trapezoid area = " + area);
```

If we start the program and enter values for the sides: 3, 4 and 5, we will obtain the following result:

```
3
4
5
Trapezoid area = 17.5
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#4.

Example: Circle Area and Perimeter

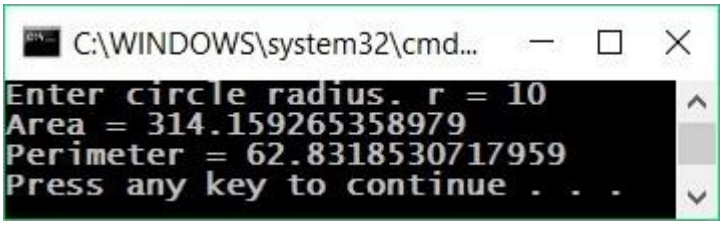
Let's write a program that calculates **a circle area and perimeter** by reading its **radius r**.

Formulas:

- Area = π * r * r
- Perimeter = 2 * π * r
- π ≈ 3.14159265358979323846…

```
Console.Write("Enter circle radius. r = ");
var r = double.Parse(Console.ReadLine());
Console.WriteLine("Area = " + Math.PI * r * r);
// Math.PI – built-in constant for π in C#
Console.WriteLine("Perimeter = " + 2 * Math.PI * r);
```

Let's test the program with **radius `r = 10`**:

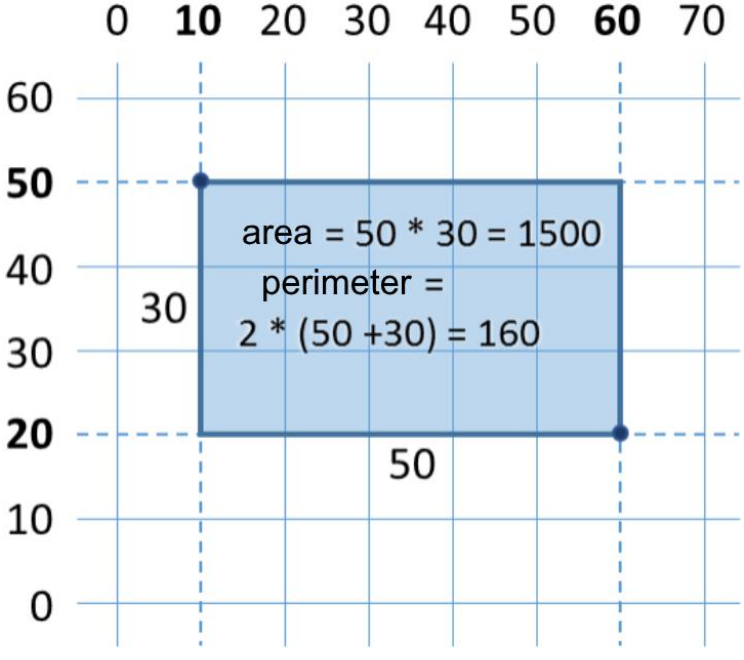


Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#5.

Example: Rectangle Area in a Coordinate Plane

A rectangle is set with the **coordinates of two of its opposite angles**. Calculate its **area and perimeter**:



In this task, we have to consider that if we subtract the smaller **x** from the bigger **x**, we will obtain the length of the rectangle. Identically, if we subtract the smaller **y** from the bigger **y**, we will obtain the height of the rectangle. What is left is to multiply both sides. Here is an example of an implementation of the described logic:

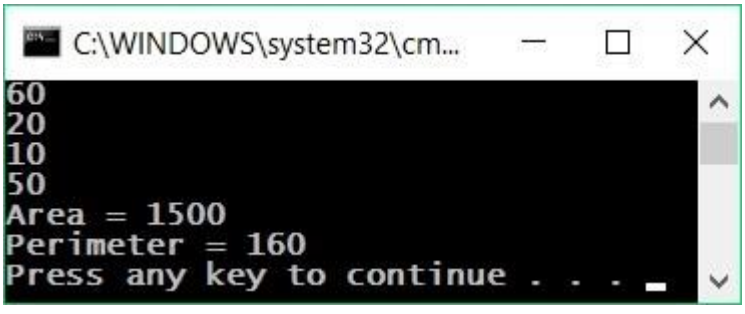
```
var x1 = double.Parse(Console.ReadLine());
var y1 = double.Parse(Console.ReadLine());
var x2 = double.Parse(Console.ReadLine());
var y2 = double.Parse(Console.ReadLine());

// Calculating the sides of the rectangle:
var width = Math.Max(x1, x2) - Math.Min(x1, x2);
var height = Math.Max(y1, y2) - Math.Min(y1, y2);
```

```
Console.WriteLine("Area = " + width * height);
Console.WriteLine("Perimeter = " + 2 * (width + height));
```

We use `Math.Max(a, b)`, to find the higher value from `a` and `b` and identically `Math.Min(a, b)` to find the lower of both values.

When the program is executed with the values from the coordinate system given in the condition, we obtain the following result:



Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#6.

Other Expressions

Expressions in C# can be not only numerical, but also can text expressions, date expressions or expressions of other type:

```
var price = 20;
var priceUSD = "$ " + price;
var priceGBP = price + " GBP";
Console.WriteLine(priceUSD); // $ 20
Console.WriteLine(priceGBP); // 20 GBP
var date = new DateTime(2017, 6, 14);
var dateAfter5days = date.AddDays(5); // 14-Jun-17
Console.WriteLine(dateAfter5days); // 19-Jun-17 00:00:00
```

Exercises: Simple Calculations

Let's strengthen the knowledge we gained with a few more exercises.

Video: Summary

Watch the following video to summarize what we learned about working with simple calculations: https://youtu.be/Zv_c-M_7Gyw.

What We Learned?

Let's summarize what we learned:

- Inserting a text: `var str = Console.ReadLine();`

- Inserting an integer: `var num = int.Parse(Console.ReadLine());`
- Inserting a float number: `var num = double.Parse(Console.ReadLine());`
- Calculations with numbers and using the suitable arithmetic operators [+, -, *, /, ()]: `var sum = 5 + 3;`
- Printing a text by placeholders on the console: `Console.WriteLine("{0} + {1} = {2}", 3, 5, 3 + 5);`

The Exercises

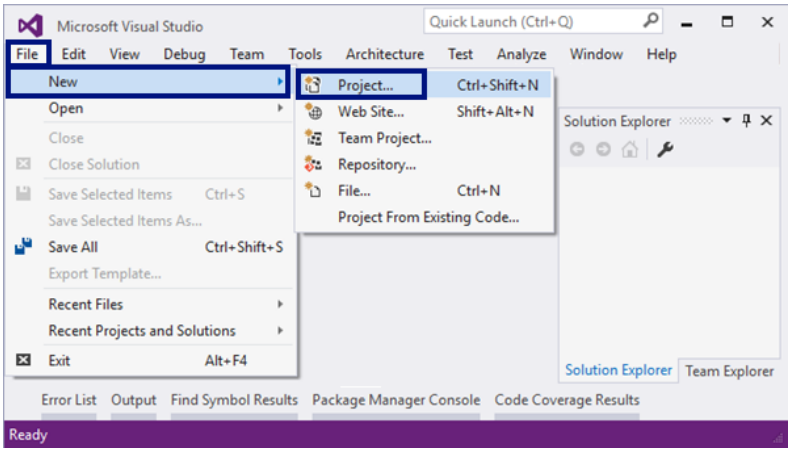
We have a lot of practical work. Solve the exercises at the end of this mini book to learn how to work with variables and data types, reading and writing on the console, using data and calculations.

Empty (Blank) Visual Studio Solution

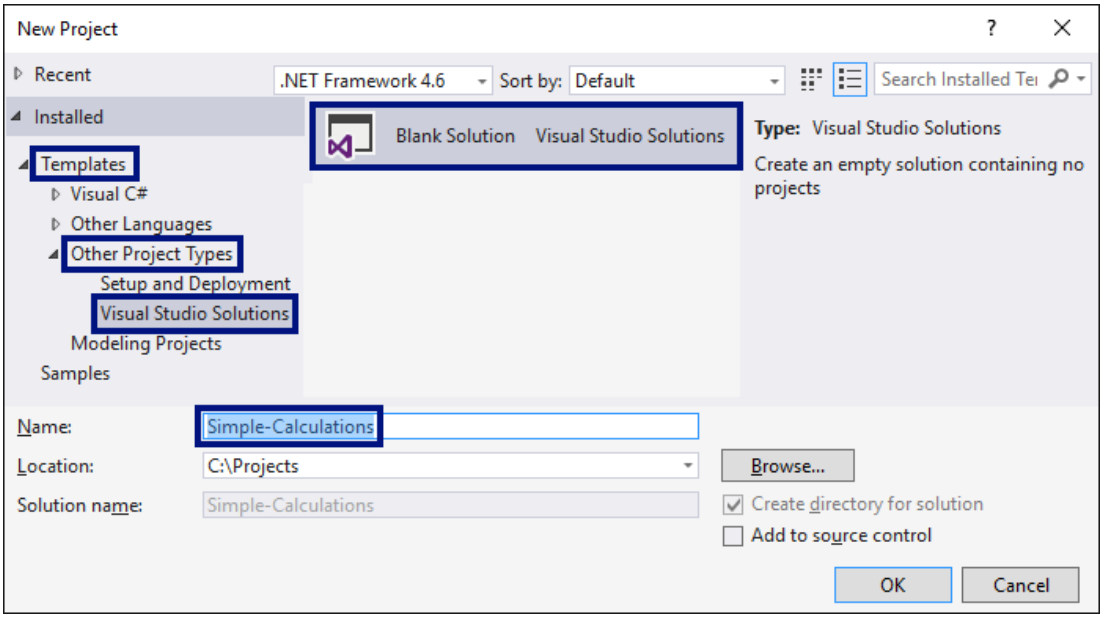
We start by creating an empty solution (Blank Solution) in Visual Studio. The solutions in Visual Studio combine a group of projects. This opportunity is very convenient, when we want to work on a few projects and switch quickly between them or we want to unite logically a few interconnected projects.

In the current practical exercise, we will use a Blank Solution with a couple of projects to organize the solutions of the tasks from the exercises – every task in a separate project and all of them in a common solution.

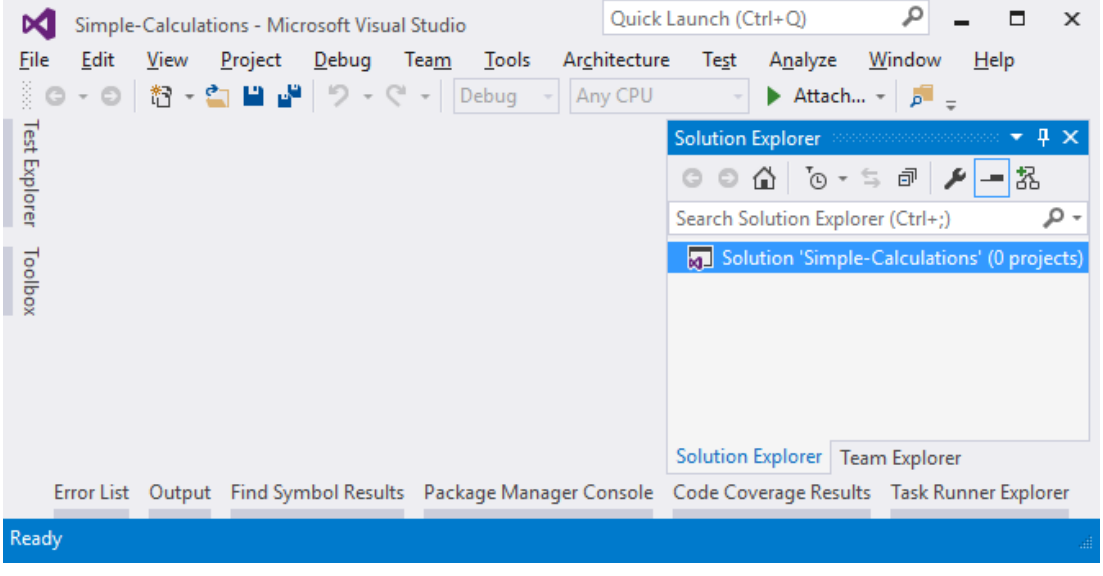
- We start Visual Studio
- We create a new Blank Solution: [File] -> [New] -> [Project].



We choose from the dialogue window [Templates] -> [Other Project Types] -> [Visual Studio Solutions] -> [Blank Solution] and we give an appropriate name of the project, for example "Simple-Calculations":



Now we have created an empty Visual Studio Solution (with 0 projects in it):



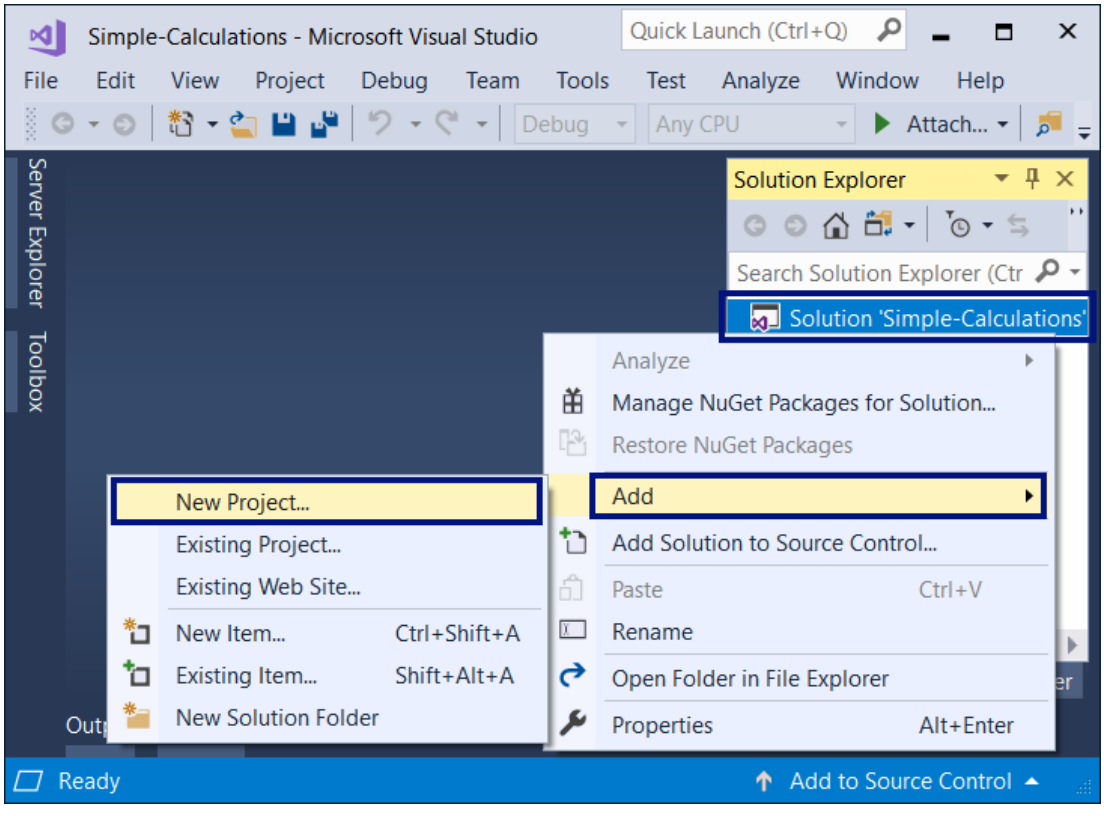
The purpose of this blank solution is to add a project per problem from the exercises.

Problem: Calculating Square Area

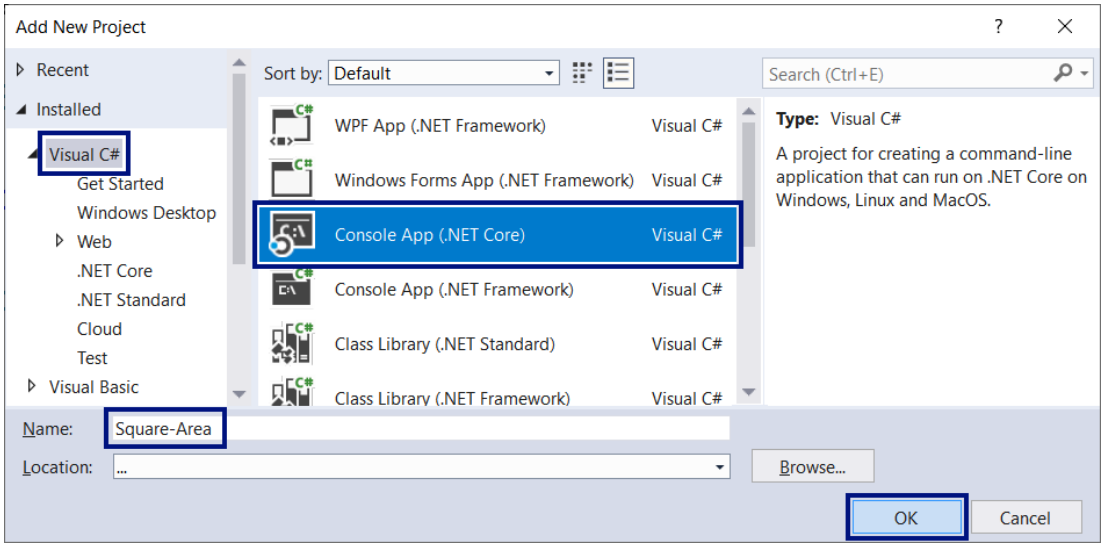
The first exercise from this topic is the following: write a console program that inputs an integer `a` and calculates the area of a square with side `a`. The task is trivial and easy: input a number from the console, multiply it by itself and print the obtained result on the console.

Hints and Guidelines

We create a new project in the existing Visual Studio solution. In the Solution Explorer right-click on Solution 'Simple-Calculations'. Choose [Add] -> [New Project...]:



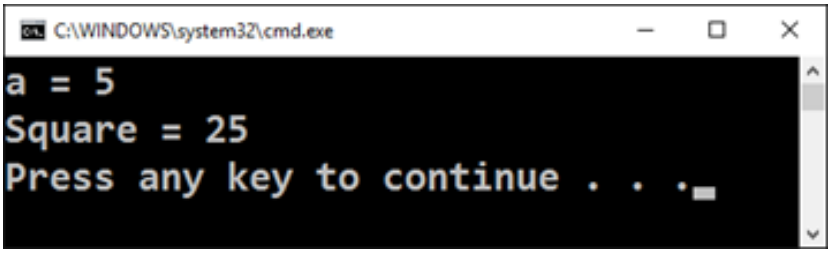
A dialogue window is going to be opened for choosing the project type for creation. We choose C# console application with name "Square-Area":



We already have a solution with one console application in it. What remains is to write the code for solving this problem. For this purpose, we go to the main method's body `Main(string[] args)` and write the following code:

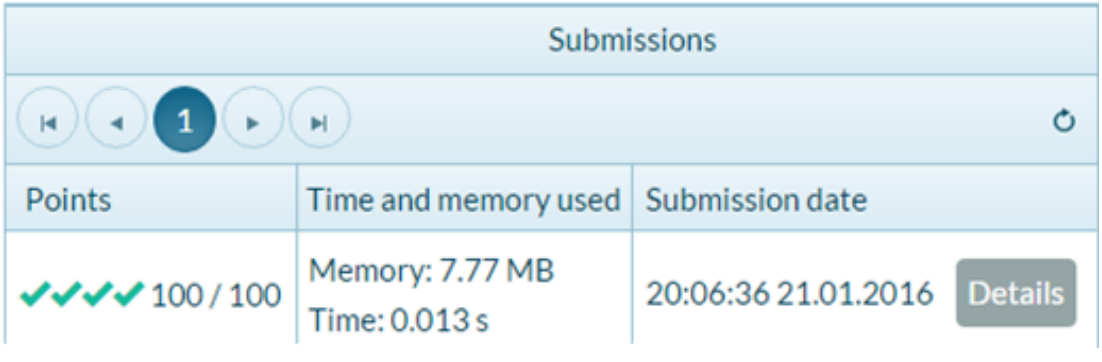
```
namespace Square_Area
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("a = ");
            var a = int.Parse(Console.ReadLine());
            var area = a * a;
            Console.Write("Square = ");
            Console.WriteLine(area);
        }
    }
}
```

The code inputs an integer through `a = int.Parse(Console.ReadLine())`, afterwards it calculates `area = a * a` and finally prints the value of the variable `area`. We start the program with [Ctrl+F5] and test it with different input values:



Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#0. You have to get 100 points (completely correct solution):

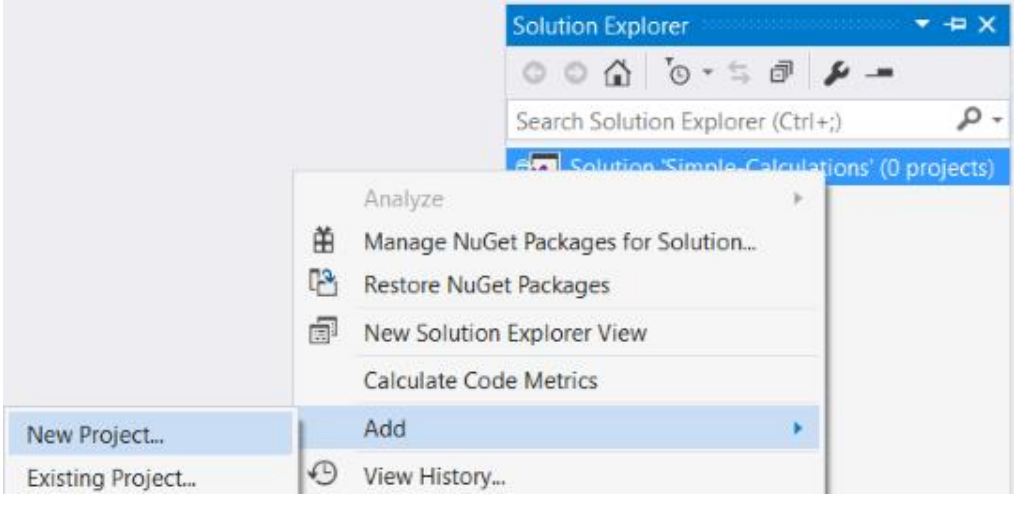


Problem: Inches to Centimeters

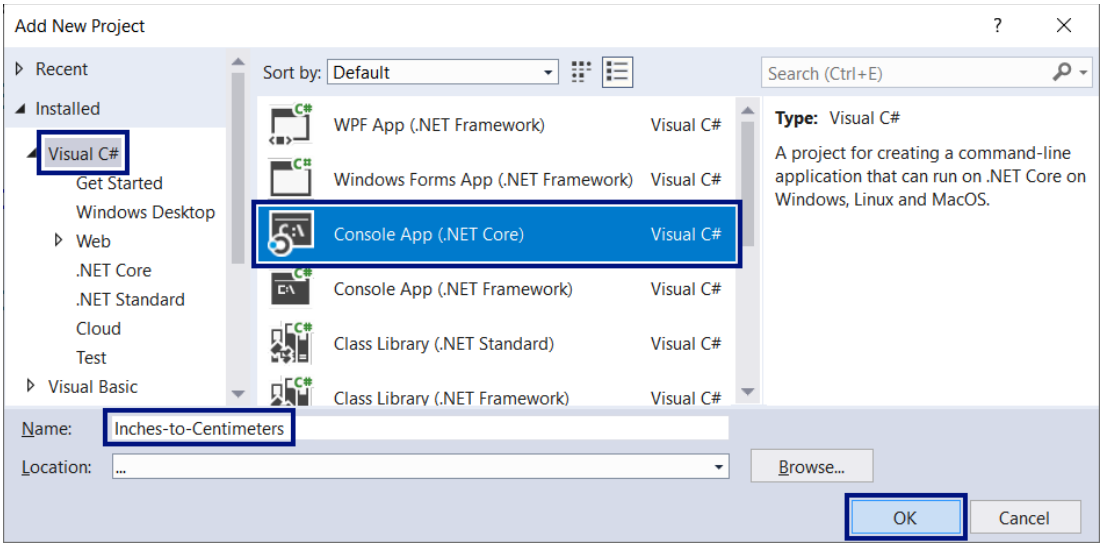
Write a program that **reads a number from the console** (not necessarily an integer) and converts the number from **inches to centimeters.** For the purpose **it multiplies the inches by 2.54** (because one inch = 2.54 centimeters).

Hints and Guidelines

First, we create a **new C# console project** in the solution "Simple-Calculations". We right-click the solution in the **Solution Explorer** and we choose [**Add**] -> [**New Project...**]:



Select [Visual C#] -> [Windows] -> [Console Application] and name it “Inches-to-Centimeters”:

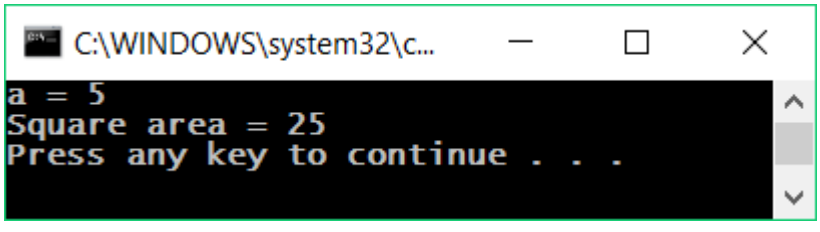


Writing Program Code and Starting the Program

Next, we have to write the program code:

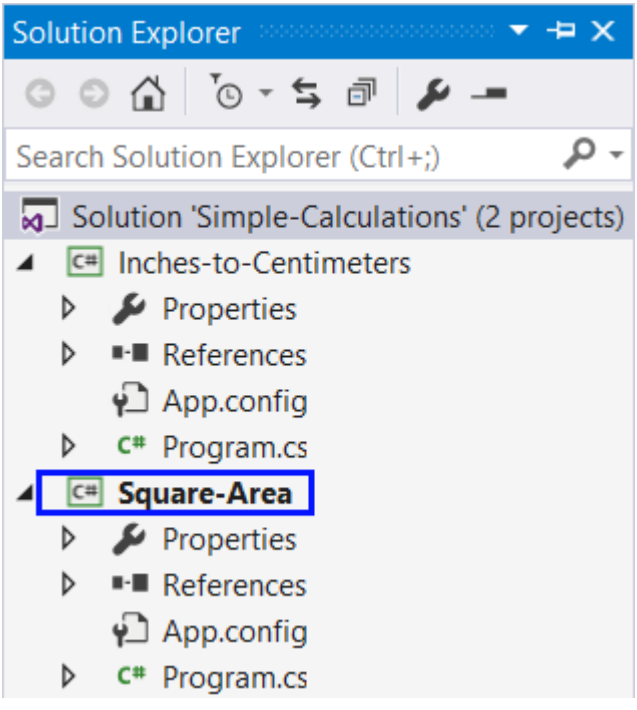
```
static void Main(string[] args)
{
    Console.Write("Inches = ");
    var inches = double.Parse(Console.ReadLine());
    var centimeters = inches * 2.54;
    Console.Write("Centimeters = ");
    Console.WriteLine(centimeters);
}
```

Start the program with [Ctrl+F5]:



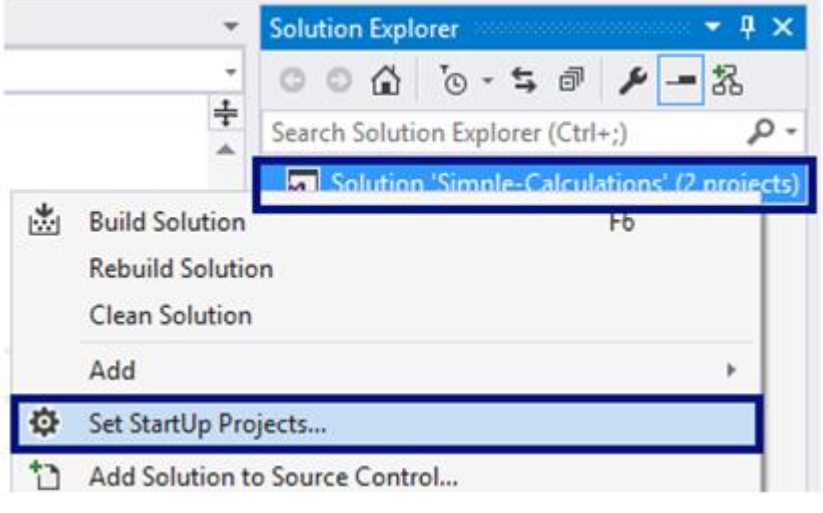
Surprise! What is happening? The program doesn't work correctly... Actually, isn't this the previous program?

In Visual Studio the current active project in a solution is marked in semi-black color and it could be changed:

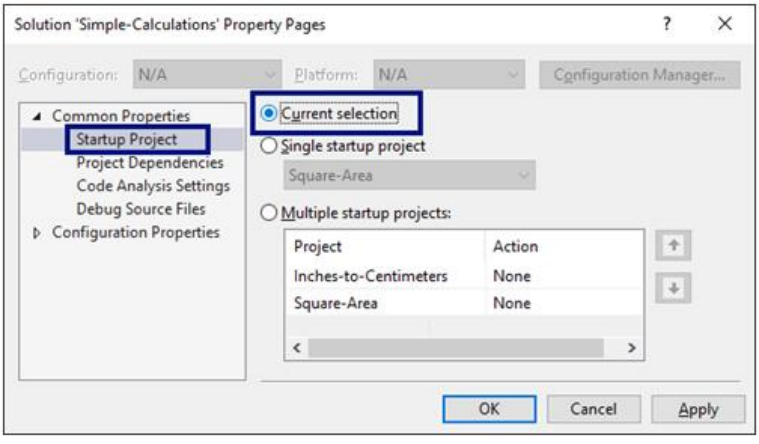


Setting up a Startup Project

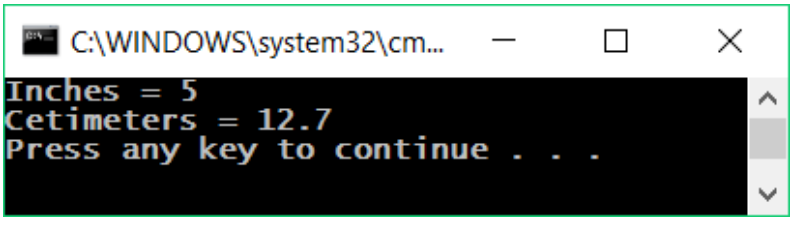
To switch the mode to **automatically go to current project**, we right-click the main solution and we choose [**Set StartUp Projects...**]:



A dialog window will open, and you will have to choose [**Startup Project**] -> [**Current Selection**]:

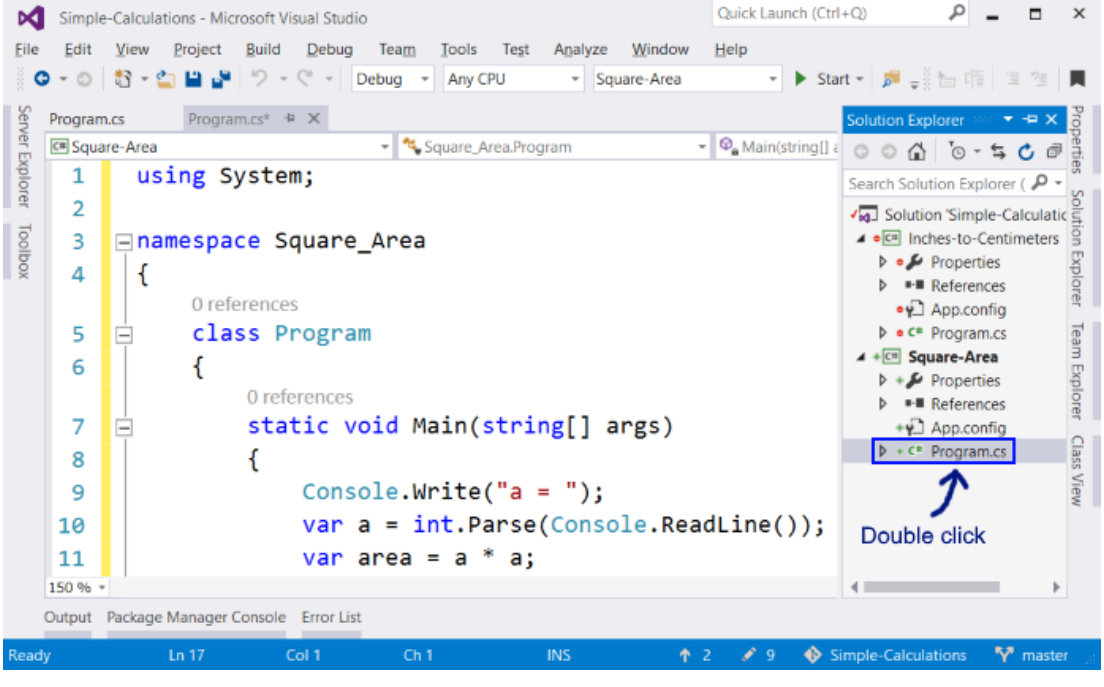


And again, **we run the program**, as usual with [Ctrl+F5]. This time it will start **the current open program**, which converts inches to centimeters. It looks like it works correctly:

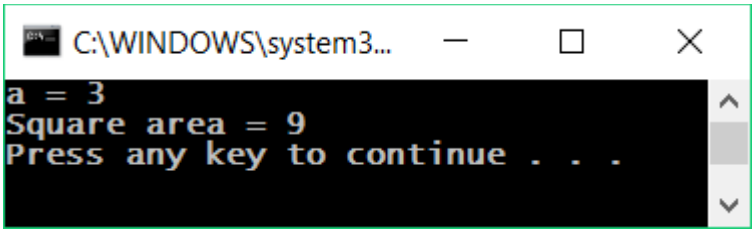


Switching Between Programs

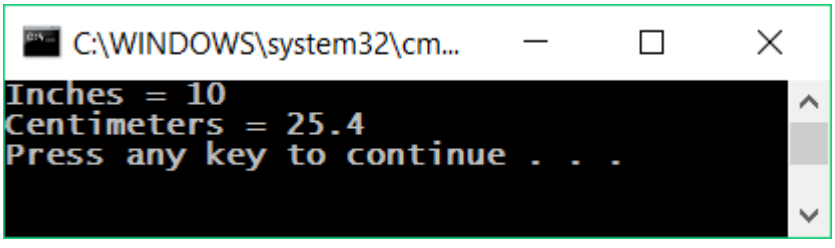
Now **let's switch to the previous program** (square area). This happens with a double click on the file `Program.cs` from the previous project **"Square-Area"** in the panel [Solution Explorer] in Visual Studio:



We press again [Ctrl+F5]. This time the other project should start:



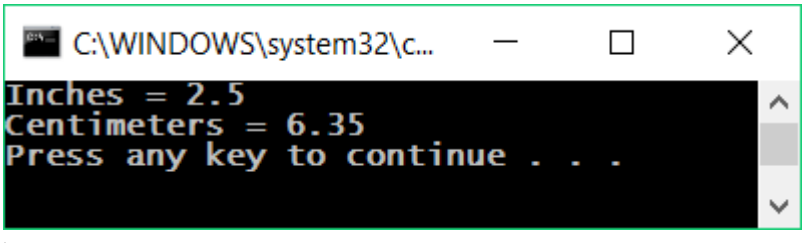
We switch back to the "Inches-to-Centimeters" project and start it with [Ctrl+F5]:



Switching between projects is very easy, isn't it? Just choose the file with the source code of the program, double click it and when it starts, the program from the current file is being executed.

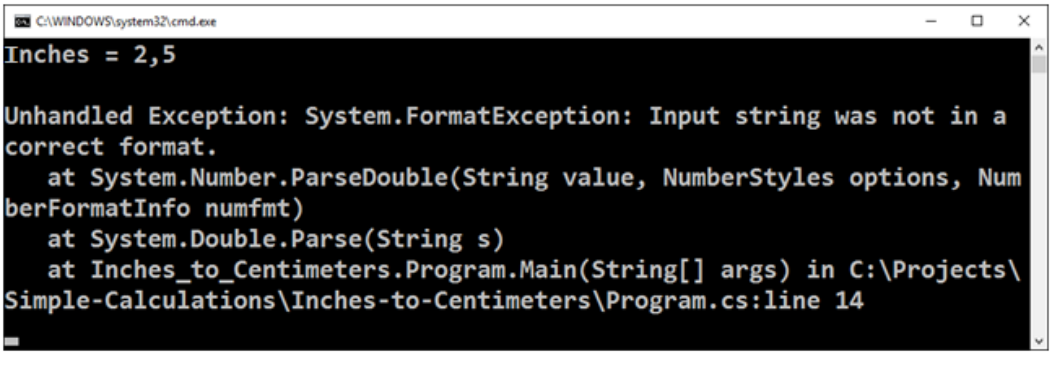
Testing a Program Locally

Let's test with floating point numbers, for example with 2.5:

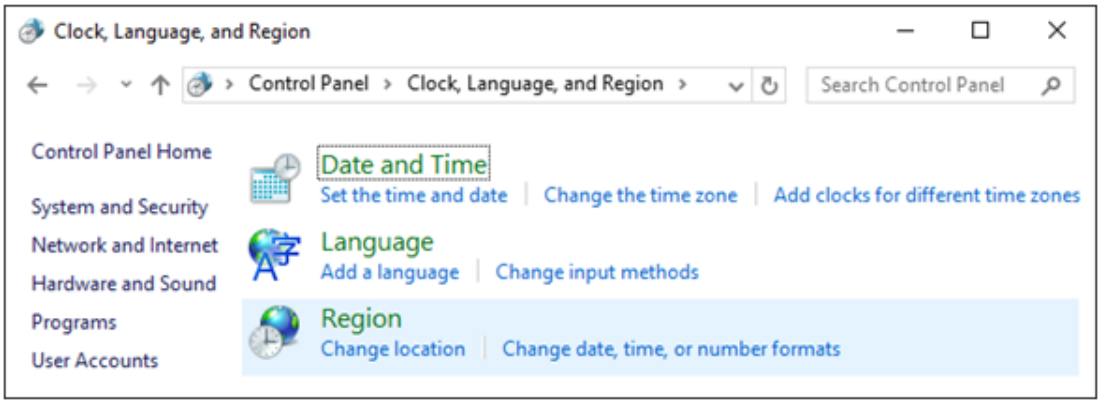


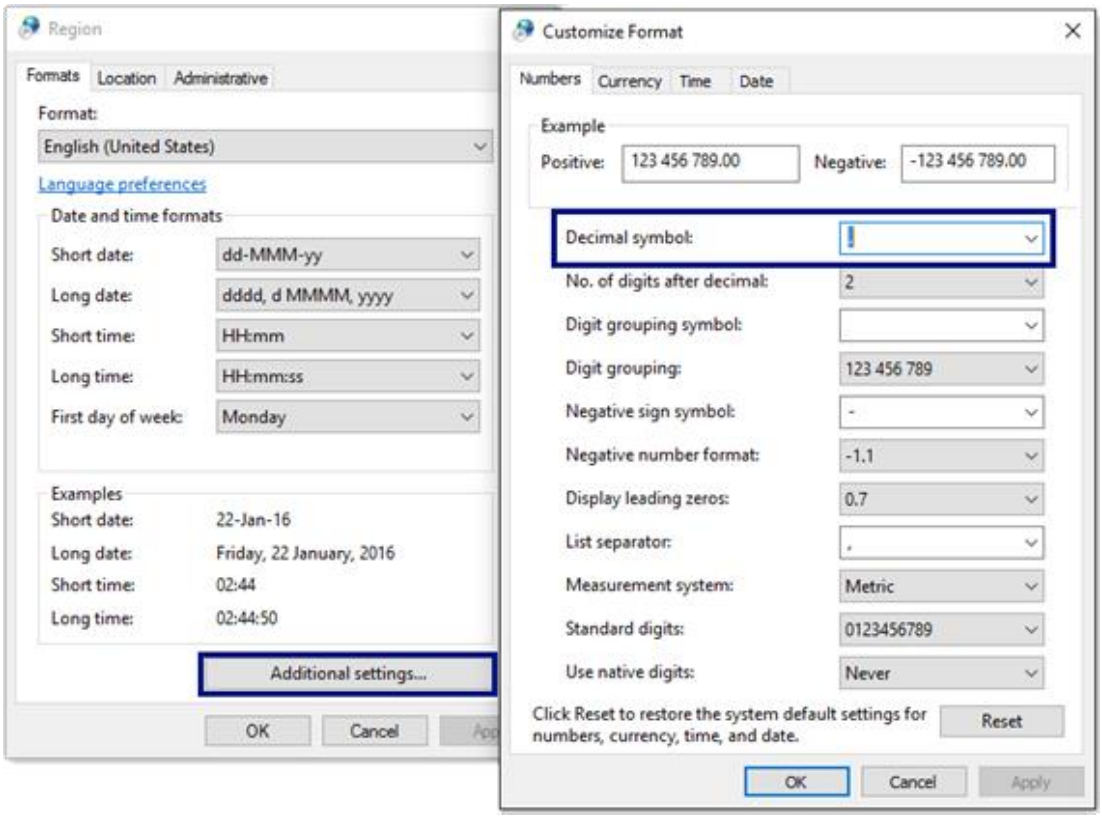
Depending on the regional settings of the operation system, it is possible instead of using a **decimal point** (US settings), to use a **decimal comma** (BG settings).

If the program expects a decimal point and instead a number with a decimal comma you enter the opposite (to enter a decimal point, when a decimal comma is expected), the following error will be produced:



It is recommended to change the settings of your computer to use a decimal point:

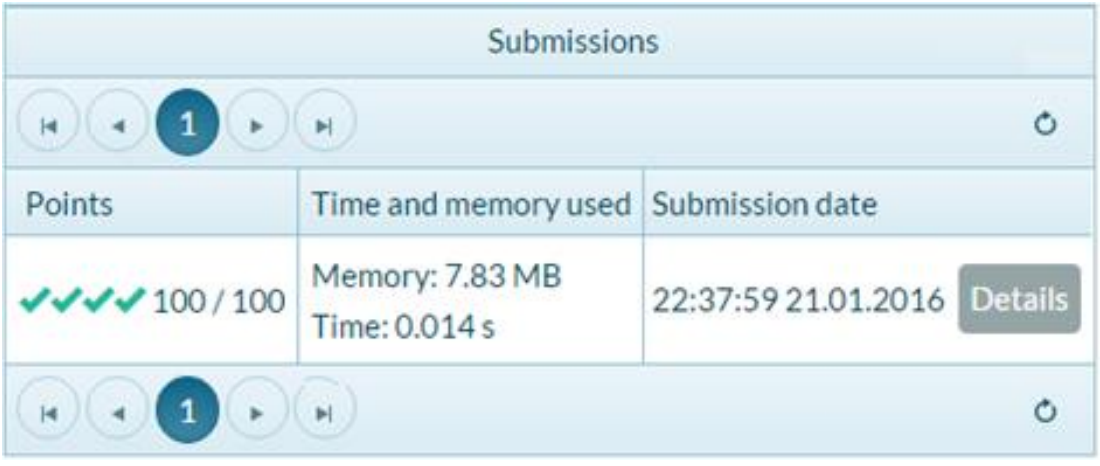




Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#1.

The solution should be accepted as a completely correct one:

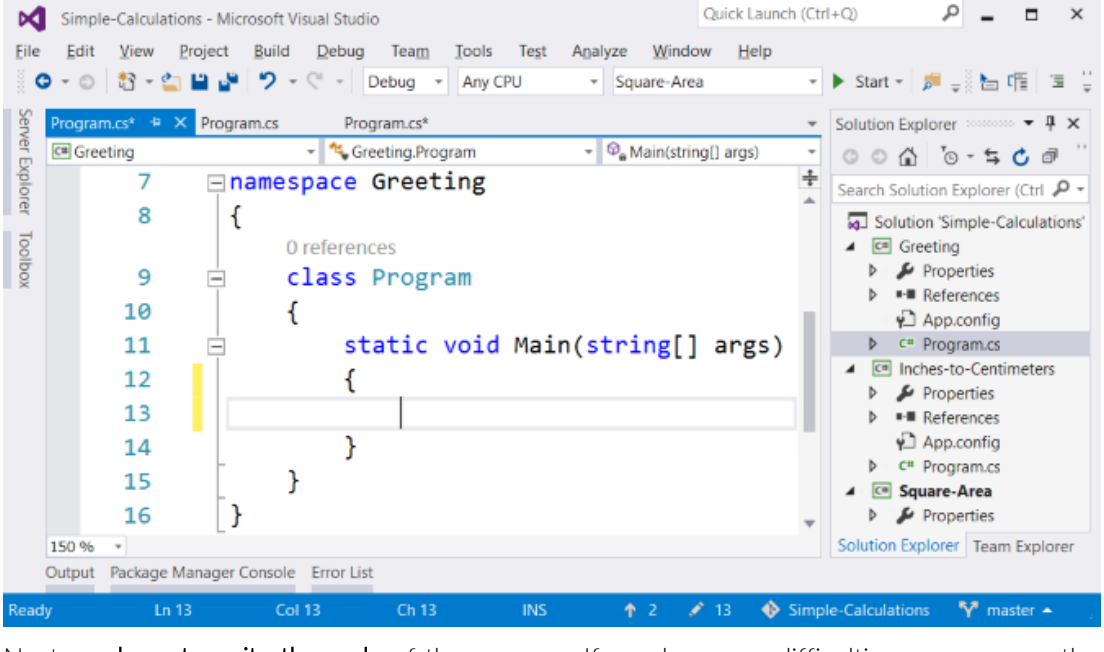


Problem: Greeting by Name

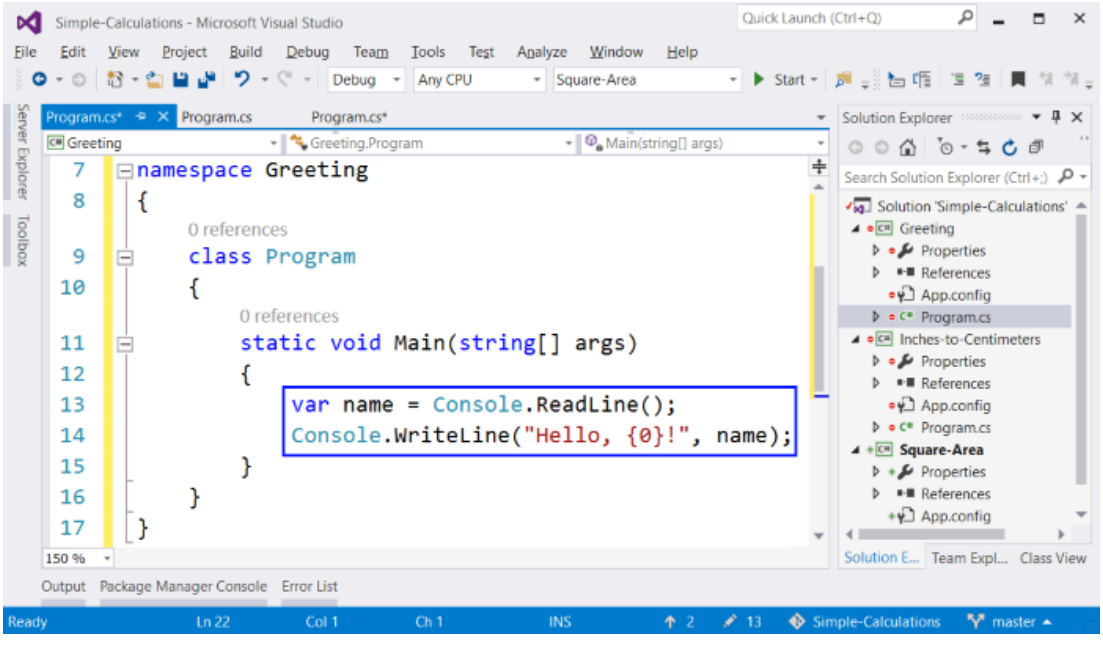
Write a program that **reads a person's name** and prints `Hello, <name>!`, where `<name>` is the name entered earlier.

Hints and Guidelines

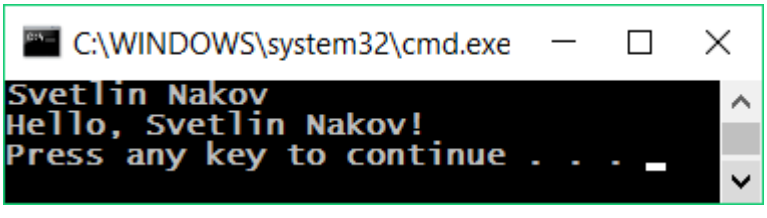
First, we create a **new C# console project** with name "Greeting" in the solution "Simple-Calculations":



Next, **we have to write the code** of the program. If you have any difficulties, you can use the code from the example below:



Run the program with [Ctrl+F5] and test if it works:



Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#2.

Problem: Concatenating Text and Numbers

Write a C# program, that reads a first name, last name, age and city from the console and prints a message of the following kind: `You are <firstName> <lastName>, a <age>-years old person from <town>`.

Hints and Guidelines

We add to the existing Visual Studio solution one more console C# project with name "Concatenate-Data". We write the code which reads the input from the console:

```
var firstName = Console.ReadLine();
var lastName = Console.ReadLine();
var age = int.Parse(Console.ReadLine());
var town = Console.ReadLine();
```

The code that prints the message described in the problem requirements should be finished.

```
Console.WriteLine(
    "You are {0} {1}, a {2}-years old person from {3}.",
    firstName, lastName, age, town);
```

In the picture above the code is blurred on purpose, in order for you to think of a way to finish it yourself.

Next, the solution should be tested locally using [Ctrl+F5] and by entering an exemplary input data.

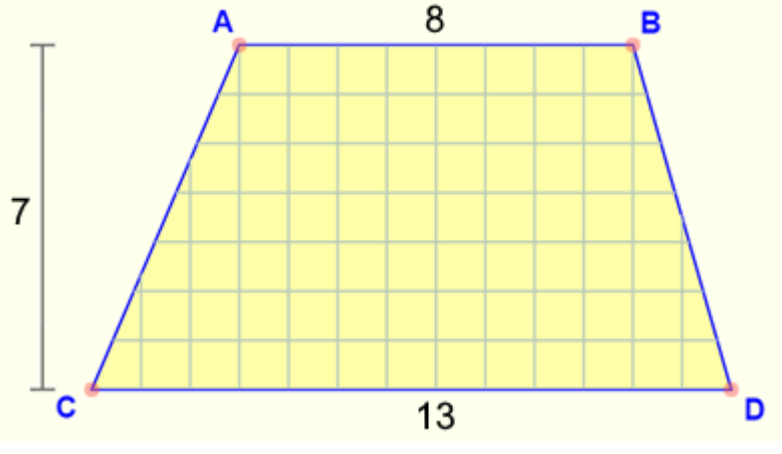
Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#3.

Problem: Trapezoid Area

Write a program that reads three numbers from the console **b1, b2 and h and calculates the area of a trapezoid** with bases **b1 and b2 and height h. The formula for trapezoid area is (b1 + b2) * h / 2**.

The figure below shows a trapezoid with bases 8 and 13 and height 7. It has an area **(8 + 13) * 7 / 2 = 73.5**.



Hints and Guidelines

Again, we have to add to the existing Visual Studio solution another **console C# project** with name "Trapezoid-Area" and write the **code that reads the input from the console, calculates the trapezoid area and prints it**:

```
static void Main(string[] args)
{
    var b1 = double.Parse(Console.ReadLine());
    var b2 = double.Parse(Console.ReadLine());
    var h = double.Parse(Console.ReadLine());
    var area = (b1 + b2) * h / 2.0;
    Console.WriteLine("Trapezoid area = " + area);
}
```

The code in the picture is purposely blurred, in order for you to give it a thought and finish it yourself.

Test your solution locally using [Ctrl+F5] and enter an exemplary data.

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#4.

Problem: Circle Area and Perimeter

Write a program that reads from the console a **number r** and calculates and prints **the area and perimeter of a circle** with **radius r**.

Input and Output

Input	Output
3	Area = 28.2743338823081 Perimeter = 18.8495559215388
4.5	Area = 63.6172512351933 Perimeter = 28.2743338823081

Video: Circle Perimeter and Area

Watch the video lesson about calculating circle perimeter and area: https://youtu.be/7W6teq9IVGU.

Hints and Guidelines

For the calculations you may use the following formulas:

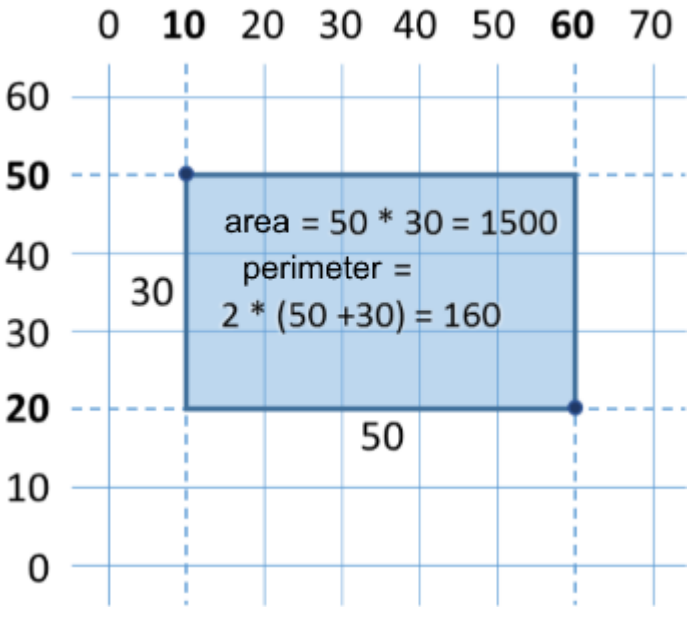
- `Area = Math.PI * r * r`.
- `Perimeter = 2 * Math.PI * r`.

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#5.

Problem: Rectangle Area

A **rectangle** is defined by the **coordinates** of both of its opposite corners (x1, y1) – (x2, y2). Calculate its **area and perimeter**. **The input** is read from the console. The numbers **x1, y1, x2 and y2** are given one per line. **The output** is printed on the console and it has to contain two lines, each with one number – the area and the perimeter.



Sample Input and Output

Input	Output
60 20 10 50	1500 160
30 40 70 -10	2000 180
600.25 500.75 100.50 -200.5	350449.6875 2402

Video: Rectangle Area

Watch the video lesson about calculating rectangle area: https://youtu.be/IHb_Tz-EVT4.

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#6.

Problem: Triangle Area

Write a program that reads from the console **a side and height of a triangle** and calculates its area. Use the **formula** for triangle area: **area = a * h / 2**. Round the result to **2 digits after the decimal point** using **`Math.Round(area, 2)`**.

Sample Input and Output

Input	Output
20 30	Triangle area = 300
15 35	Triangle area = 262.5
7.75 8.45	Triangle area = 32.74
1.23456 4.56789	Triangle area = 2.82

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#7.

Problem: Console Converter – from °C Degrees to °F Degrees

Write a program that reads **degrees on Celsius scale** (°C) and converts them to **degrees on Fahrenheit scale** (°F). Look on the Internet for a proper formula to do the calculations. Round the result to **2 digits after the decimal point**. Here are a few examples:

Sample Input and Output

Input	Output
25	77
0	32
-5.5	22.1
32.3	90.14

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#8.

Problem: Console Converter – from Radians to Degrees

Write a program, that reads **an angle in radians** (`rad`) and converts it to degrees (`deg`). Look for a proper formula on the Internet. The number **π** in C# programs is available through `Math.PI`. Round the result to the nearest integer using the `Math.Round(…)` method.

Sample Input and Output

Input	Output
3.1416	180
6.2832	360
0.7854	45
0.5236	30

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#9.

Problem: Console Converter – USD to BGN

Write a program for **conversion of US dollars** (USD) **into Bulgarian levs** (BGN). **Round** the result to **2 digits** after the decimal point. Use a fixed rate between a dollar and lev: **1 USD = 1.79549 BGN**.

Sample Input and Output

Input	Output
20	35.91 BGN
100	179.55 BGN
12.5	22.44 BGN

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#10.

Problem: * Console Currency Converter

Write a program for **conversion of money from one currency into another**. It has to support the following currencies: **BGN, USD, EUR, GBP**. Use the following fixed currency rates:

Rate	USD	EUR	GBP
1 BGN	1.79549	1.95583	2.53405

The input is **sum for conversion**, **input currency** and **output currency**. **The output** is one number – the converted value of the above currency rates, rounded **2 digits** after the decimal point.

Sample Input and Output

Input	Output
20 USD BGN	35.91 BGN
100 BGN EUR	51.13 EUR
12.35 EUR GBP	9.53 GBP
150.35 USD EUR	138.02 EUR

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#11.

Problem: ** Date Calculations – 1000 Days on the Earth

Write a program that enters **a birth date** in format `dd-MM-yyyy` and calculates the date on which **1000 days** are turned since this birth date and prints it in the same format.

Sample Input and Output

Input	Output
25-02-1995	20-11-1997
07-11-2003	02-08-2006
30-12-2002	24-09-2005
01-01-2012	26-09-2014
14-06-1980	10-03-1983

Hints and Guidelines

- Look for information about the data type `DateTime` in C# and in particular look at the methods `ParseExact(str, format)`, `AddDays(count)` and `ToString(format)`. With their help you can solve the problem without the need to calculate days, months and leap years.
- **Don't print** anything additional on the console except for the wanted date!

Testing in the Judge System

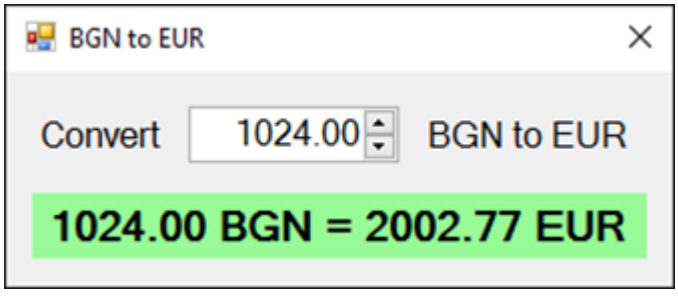
Test your solution here: https://judge.softuni.org/Contests/Practice/Index/504#12.

Lab: GUI Applications with Numerical Expressions

To exercise working with variables and calculations with operators and numerical expressions, we will make something interesting: we will develop a **desktop application** with graphical user interface. In it, we will use calculations with floating point numbers.

Graphical Application: Converter from BGN to EUR

We need to create **a graphical application** (GUI application) that calculates the value in **Euro** (EUR) of monetary amount given in **leva** (BGN). By changing the amount in leva, the amount in Euro has to be recalculated automatically (we use rate leva / Euro: **1.95583**).



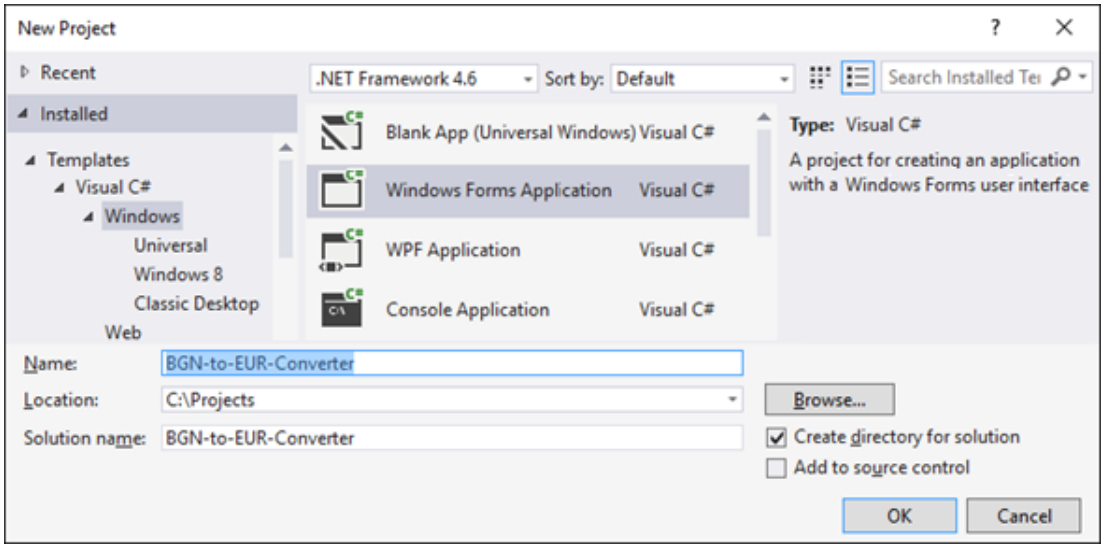
This exercise goes beyond the material learned in this book and its purpose is not to teach you how to program GUI applications, but to strengthen your interest in software development. Let's get to work.

Video: GUI Converter from BGN to EUR

Watch the following video lesson to learn how to build the Windows Forms based GUI application to convert BGN to EUR: https://youtu.be/xWbDjzLsu9U.

Creating a New C# Project

We add to the existing Visual Studio solution one more project. This time we create a **Windows Forms** application with C# named "**BGN-to-EUR-Converter**":

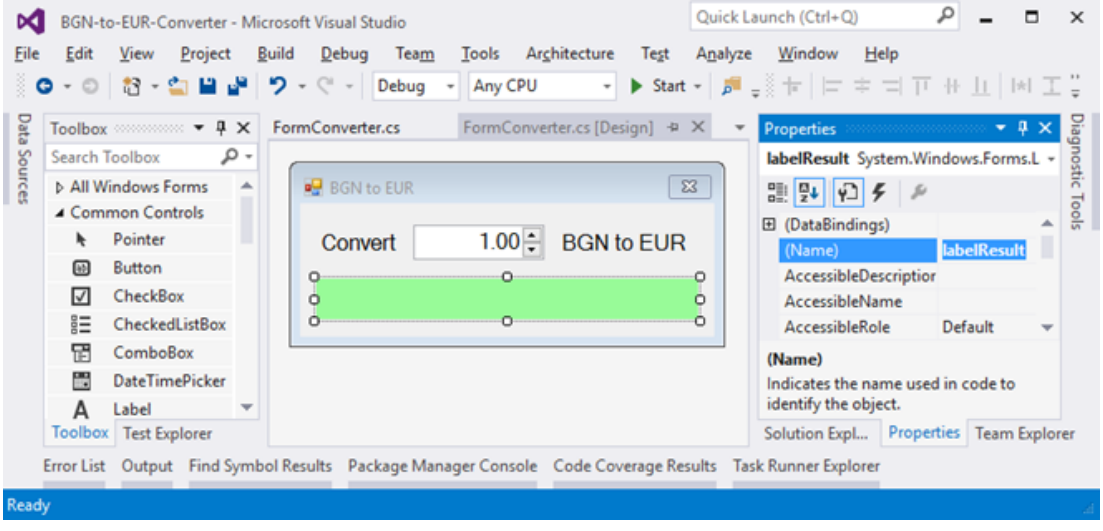


Adding UI Controls

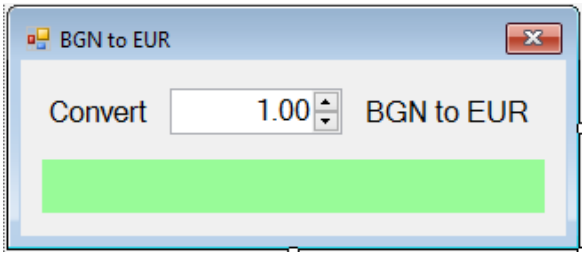
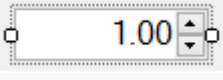
We arrange the following UI (user interface) controls in the format:

- **`NumericUpDown`** with name **`numericUpDownAmount`** – it will enter the amount for conversion
- **`Label`** with name **`labelResult`** – it will show the result after conversion
- Two more **`Label`** components, serving only for static representation of a text

The graphical editor for user interface might look similar to this:



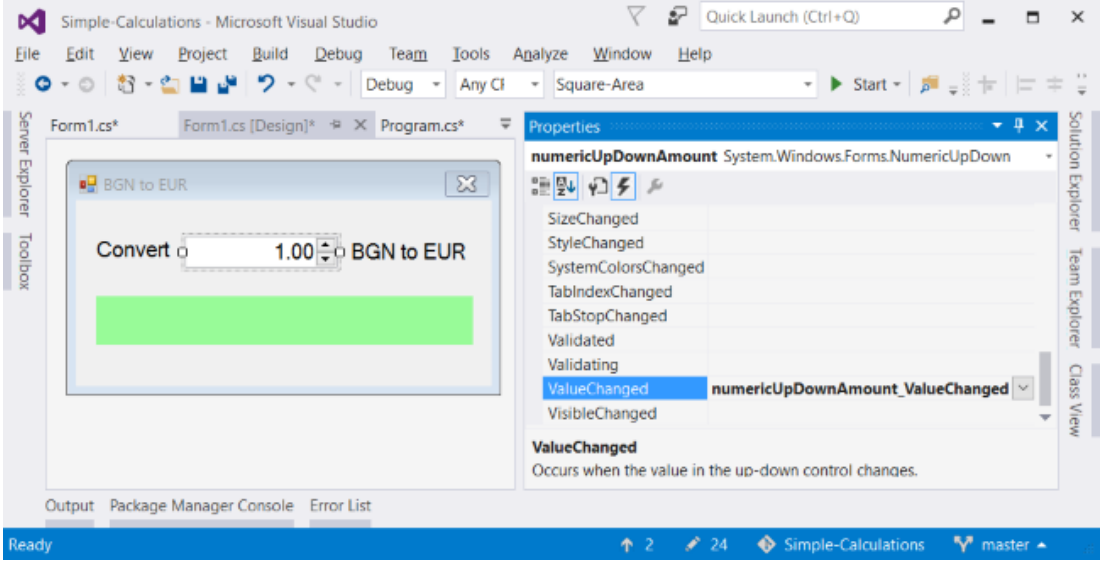
We do the following settings of the format and the separate controls:

Setting	Picture
`FormConverter`: Text = "BGN to EUR", Font.Size = 12, MaximizeBox = False, MinimizeBox = False, FormBorderStyle = FixedSingle	
`numericUpDownAmount`: Value = 1, Minimum = 0, Maximum = 10000000, TextAlign = Right, DecimalPlaces = 2	
`labelResult`: AutoSize = False, BackColor = PaleGreen, TextAlign = MiddleCenter,	

Setting	Picture
Font.Size = 14, Font.Bold = True	

Events and Event Handlers

We define the following event handlers in the controls:

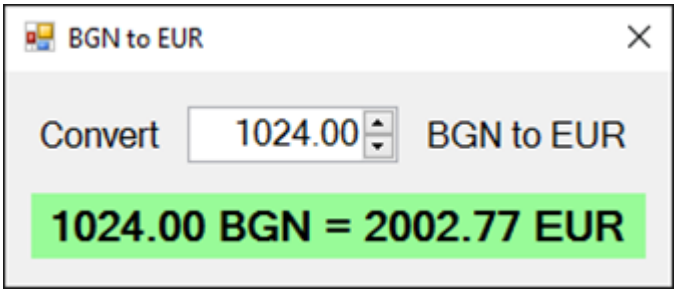


After this we catch the following events:

- **`FormConverter.Load`** (by double-clicking with the mouse)
- **`numericUpDownAmount.ValueChanged`** (by double-clicking on **`NumericUpDown`** control)
- **`numericUpDownAmount.KeyUp`** (we choose **`Events`** from the dashboard **`Properties`** and double-click on **`KeyUp`**)

The event **`Form.Load`** is executed when the program is started, before the window of the application is shown. The event **`NumericUpDown.ValueChanged`** is executed when a change in the value inside the field for entering a number occurs. The event **`NumericUpDown.KeyUp`** is executed after pressing a key in the field that enters a number. On the occurrence of each of these events, we will recalculate the result.

To catch an event, we use the events icon (Events) in the [Properties] window in Visual Studio:



Writing the Program Code

We will use the following C# code for handling events:

```
private void FormConverter_Load(object sender, EventArgs e)
{
  ConvertCurrency();
}

private void numericUpDownAmount_ValueChanged(object sender, EventArgs
e)
{
  ConvertCurrency();
}

private void numericUpDownAmount_KeyUp(object sender, KeyEventArgs e)
{
  ConvertCurrency();
}
```

All of the caught events call the method **`ConvertCurrency()`**, which converts the given sum from leva to Euro and shows the result in the green box.

We have to write the code (program logic) for converting from leva to Euro:

```
private void ConvertCurrency()
{
  var amountBGN = this.numericUpDownAmount.Value;
  var amountEUR = amountBGN * 1.95583m;
  this.labelResult.Text =
    amountBGN + " BGN = " +
    Math.Round(amountEUR, 2) + " EUR";
}
```

Testing the Application

Finally, we start the project with [Ctrl+F5] and test if it works correctly.

Graphical Application: * Catch the Button!

Create a fun graphical application "catch the button": a form consisting of one button. Upon moving the mouse cursor onto the button, it moves to a random position. This way it creates the impression that "the button runs form the mouse and it is hard to catch". When the button gets "caught", a congratulations message is shown.

Hints and Guidelines

Write an event handler `Button.MouseEnter` and move the button to a random position. Use the random numbers generator `Random`. The position of the button is set using the `Location` property. To make sure the new position of the button is within the form's borders, you can make calculations based on the size of the form, which is available from the `ClientSize` property.

You may use the following sample code:

```
private void buttonCatchMe_MouseEnter(object sender, EventArgs e)
{
    Random rand = new Random();
    var maxWidth = this.ClientSize.Width -
buttonCatchMe.ClientSize.Width;
    var maxHeight = this.ClientSize.Height -
buttonCatchMe.ClientSize.Height;
    this.buttonCatchMe.Location = new Point(
        rand.Next(maxWidth), rand.Next(maxHeight));
}
```

What's Next?

The "Programming Basics" series consists of publications for beginners in programming that covers the following topics:

1. First Steps in Programming – commands, programs, C#, Visual Studio,
2. Simple Calculations – variables, calculations, console input / output
3. Simple Conditions – conditional statements, the "if-else" construction
4. More Complex Conditions – nested if-else, logical "and", "or", "not"
5. Repetitions (Loops) – simple for-loops (repeat from 1 to n)
6. Nested Loops – nested loops and problem solving, drawing 2D figures
7. More Complex Loops – loops with a step, infinite loops with breaks
8. How to Become a Software Engineer?

The next topics are coming. Be patient!

Sign-Up to Study in SoftUni

The easiest way to become a software engineer is to go through the "Software University" training program at SoftUni. Signup now for the free Programming Basics training course:

https://softuni.org/apply

Join the SoftUni community and study programming for free in our interactive training platform. Get live support and mentoring from our trainers. Become a software developer now!