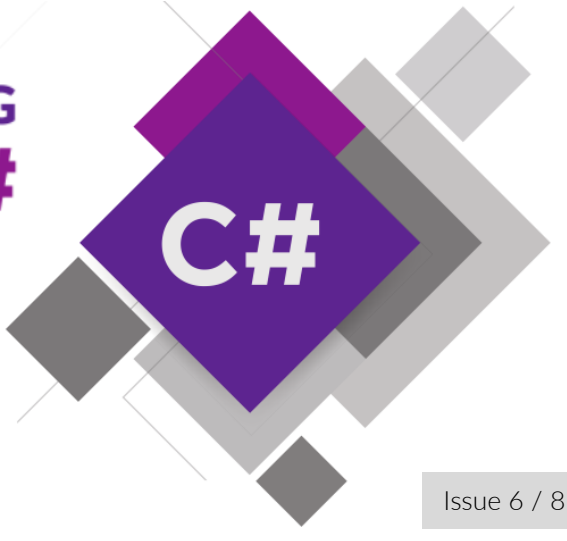
PROGRAMMING BASICS WITH C#



Issue 6 / 8

NESTED LOOPS

Learn about nesting loops one inside another and how to use nested loops and conditional statements to design more complex program logic, e.g. generate and draw tables, draw 2D figures on the console and others.

Get an idea how to build simple Web application "Ratings" in C# and ASP.NET MVC using Visual Studio.

Dr. Svetlin Nakov

Software University – https://softuni.org

Nested Loops

In this mini book we will be looking at **nested loops** and how to use `for` loops to **draw** various **figures on the console**, that contain symbols and signs, ordered in rows and columns on the console. We will use **single** and **nested loops** (loops that stay in other loops), **calculations** and **checks**, in order to print on the console simple and not so simple figures by specified sizes.

Video: Overview

Watch a video lesson to learn what we will learn: https://youtu.be/sbmhyr1Yz7U.

Introduction to Nested Loops by Examples

In programming **loops can be nested**, which means that in a loop we can put another loop. This is an example of **nested for-loops**, which are used to draw a square of `n` rows, each holding `n` times the chars `=-`:

```
int n = int.Parse(Console.ReadLine());
for (int row = 1; row <= n; row++)
{
    for (int col = 1; col <= n; col++)
    {
        Console.Write("=-");
    }
    Console.WriteLine();
}
```

Run the above code example: https://repl.it/@nakov/nested-for-loops-draw-square-csharp.

If we run the above code and enter `5` as input, the **output** will be as follows:

```
5
=-=-=-=-=-
=-=-=-=-=-
=-=-=-=-=-
=-=-=-=-=-
=-=-=-=-=-
```

Using a combination of **calculations**, **conditional statements** and **nested loops**, we can implement **more complex logic**. For example, we can draw a rhombus of stars as follows:

```
int n = 10;
for (int row = 1; row < n; row++)
{
    var spaces = Math.Abs(n / 2 - row);
    var stars = n/2 - spaces;
    for (int col = 1; col <= spaces; col++)
        Console.Write(" ");
    for (int col = 1; col <= stars; col++)
        Console.Write("* ");
```

```
        Console.WriteLine();
    }
```

Run the above code example: https://repl.it/@nakov/nested-for-loops-draw-rhombus-csharp.

The above code will print on the console the following output:

```
     *
    * *
   * * *
  * * * *
 * * * * *
  * * * *
   * * *
    * *
     *
```

Let's explain in greater detail how to use nested loops to implement more complex login in our C# programs.

Nested Loops – Concepts

A nested loop is a construction where in the body of one loop (outer one) stays another loop (inner one). In each iteration of the outer loop, the whole inner loop is executed. This happens in the following way:

- When nested loops start executing, the outer loop starts first: the controlling variable is initialized and after a check for ending the loop the code in its body is executed.
- After that, the inner loop is executed. The controlling variables start position is initialized, a check for ending the loop is made and the code in its body is executed.
- When reaching the specified value for ending the loop, the program goes back one step up and continues executing the previous (outer) loop. The controlling variable of the outer loop changes with one step, a check is made to see if the condition for ending the loop is met and a new execution of the nested (inner) loop is started.
- This is repeated until the variable of the outer loop meets the condition to end the loop.

Video: Nested Loops

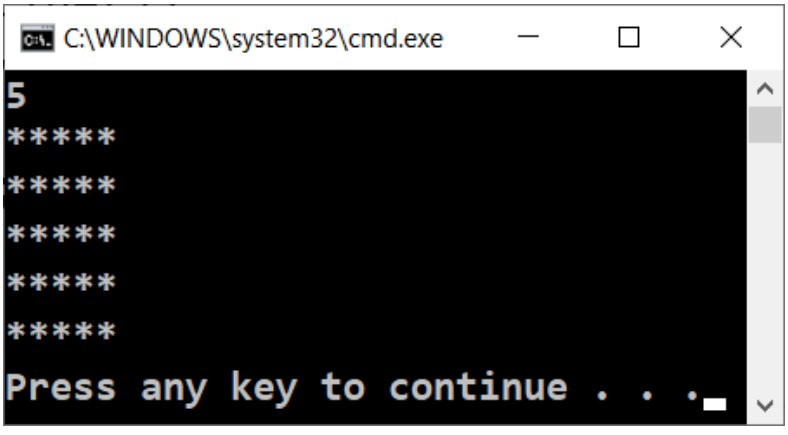
Watch this video to learn the concepts of nested loops: https://youtu.be/Sj49u6XlzXU.

Nested Loops – Examples

Here is an example that illustrates nested loops. The aim is again to print a rectangle made of `n * n` stars, in which for each row a loop iterates from `1` to `n`, and for each column a nested loop is executed from `1` to `*n`:

```
var n = int.Parse(Console.ReadLine());
for (var r = 1; r <= n; r++)
{
    for (var c = 1; c <= n; c++)
    {
        Console.Write("*");
    }
    Console.WriteLine();
}
```

If we enter **5** as input on the console, the above sample code will print the following output:



Let's look at the example above. After initializing **the first (outer) loop**, its **body**, which contains **the second (nested) loop** starts executing. By itself it prints on one row **n** number of stars. After **the inner** loop **finishes** executing at the first iteration of the outer one, **the first loop will continue**, i.e. it will print an empty row on the console. **After that**, the variable of **the first** loop will be **renewed** and the whole **second** loop will be executed again. The inner loop will execute as many times as the body of the outer loop executes, in this case **n** times.

Example: Rectangle Made of 10 x 10 Stars

Print on the console a rectangle made out of **10 x 10** stars.

Input	Output
(none)	********** ********** ********** ********** ********** ********** ********** ********** ********** **********

Video: Rectangle of 10 x 10 Stars

Watch this video lesson to learn how to print a rectangle of 10 x 10 stars on the console using nested for-loops: https://youtu.be/XNsgT4yqw_s.

Hints and Guidelines

```
for (var i = 1; i <= 10; i++)
{
    Console.WriteLine(new string('*', 10));
}
```

How does the example work? We initialize a loop with a variable `i = 1`, which increases with each iteration of the loop, while it is less or equal to 10. This way the code in the body of the loop is executed 10 times. In the body of the loop we print a new line on the console **`new string('*', 10)`**, which creates a string of 10 stars.

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/512#0.

Example: Rectangle Made of N x N Stars

Write a program that gets a positive integer n and prints on the console a rectangle made out of N x N stars.

Input	Output	Input	Output	Input	Output
2	`**` `**`	3	`***` `***` `***`	4	`****` `****` `****` `****`

Video: Rectangle of N x N Stars

Watch this video lesson to learn how to print a rectangle of 10 x 10 stars on the console using nested for-loops: https://youtu.be/9sB4Z2TI1AE.

Hints and Guidelines

```
int n = int.Parse(Console.ReadLine());
for (int i = 1; i <= n; i++)
{
    Console.WriteLine(
        new string('*', n));
}
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/512#1.

Examples: Square Made of Stars

Print on the console a square made of N x N stars (use a space between the stars, staying on the same line):

Input	Output	Input	Output	Input	Output
2	* * * *	3	* * * * * * * * *	4	* * * * * * * * * * * * * * * *

Hints and Guidelines

The problem is similar to the last one. The difference here is that we need to figure out how to add a whitespace after the stars so that there aren't any excess white spaces in the beginning or the end.

```
var n = int.Parse(Console.ReadLine());
for (var r = 1; r <= n; r++)
{
    Console.Write("*");
    for (var c = 1; c < n; c++)
    {
        Console.WriteLine(" *");
    }
    Console.WriteLine();
}
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/512#2.

Example: Triangle Made of Dollars

Write a program that takes an integer n and prints a triangle made of dollars of size n.

Input	Output	Input	Output	Input	Output
3	$ $ $ $ $ $	4	$ $ $ $ $ $ $ $ $ $	5	$ $ $ $ $ $ $ $ $ $ $ $ $ $ $

Video: Triangle of Dollars

Watch this video to learn how to print a triangle of dollars on the console, using nested for-loops: https://youtu.be/Pbfe1F0nMNE.

Hints and Guidelines

The problem is similar to those for drawing a rectangle and square. Once again, we will use nested loops, but there is a catch here. The difference is that the number of columns that we need to print depends on the row, on which we are and not on the input number **n**. From the example input and output data we see that the count of dollars depends on which row we are on at the moment of the printing, i.e. 1 dollar means first row, 3 dollars mean third row and so on. Let's see the following example in detail. We see that the variable of the nested loop is connected with the variable of the outer one. This way our program prints the desired triangle.

```
var n = int.Parse(Console.ReadLine());
for (var row = 1; row <= n; row++)
{
    Console.Write("$");
    for (var col = 1; col < row; col++)
    {
        Console.Write(" $");
    }
    Console.WriteLine();
}
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/512#3.

Example: Square Frame

Write a program that takes a positive integer n and draws on the console a square frame with a size of n * n.

Input	Output	Input	Output
3	+ - + | - | + - +	4	+ - - + | - - | | - - | + - - +

Input	Output	Input	Output
5	+ - - - + | - - - | | - - - | | - - - | + - - - +	6	+ - - - - + | - - - - | | - - - - | | - - - - | | - - - - | + - - - - +

Video: Square Frame

Watch this video lesson to learn how to print a square frame on the console using nested loops: https://youtu.be/LS2uqvggfSA.

Hints and Guidelines

We can solve the problem in the following way:

- We read from the console the number **n**.
- We print the upper part: first a **+** sign, then n-2 times **-** and in the end a **+** sign.
- We print the middle part: we print n-2 rows, as we first print a **|** sign, then n-2 times **-** and in the end again a **|** sign. We can do this with nested loops.
- We print the lower part: first a **+** sign, then n-2 times **-** and in the end a **+** sign.

Here is an example implementation of the above idea with nested loops:

```
int n = int.Parse(Console.ReadLine());

// Print the top row: + - - - +
Console.Write("+");
for (int i = 0; i < n - 2; i++)
{
    Console.Write(" -");
}
Console.WriteLine(" +");

// Print the mid rows: | - - - |
for (int row = 0; row < n - 2; row++)
{
    Console.Write("|");
    for (int i = 0; i < n - 2; i++)
    {
        Console.Write(" -");
    }
    Console.WriteLine(" |");
}

// Print the bottom row: + - - - +
Console.Write("+");
for (int i = 0; i < n - 2; i++)
{
    Console.Write(" -");
}
Console.WriteLine(" +");
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/512#4.

Exercises: Drawing Figures

Let's look at how to **draw figures** using **nested loops** with more complex logic, for which we need to think more before coding.

Video: Summary

Watch this video to review what we learned: https://youtu.be/tj4s3BOijO4.

What We Learned?

Before starting, let's review what we learned.

We became acquainted with the `new string` constructor:

```
string printMe = new string('*', 5);
```

We learned to draw figures with nested `for` loops:

```
for (var r = 1; r <= 5; r++)
{
   Console.Write("*");
   for (var c = 1; c < 5; c++)
      Console.Write(" *");
   Console.WriteLine();
}
```

Problem: Rhombus Made of Stars

Write a program that takes a positive integer **n** and prints **a rhombus made of stars** with size **n**.

Input	Output	Input	Output

1	`*`	2	` *`
			`* *`
			` *`

Input	Output	Input	Output

3	`  *`
	` * *`
	`* * *`
	` * *`
	`  *`
		4	`   *`
			`  * *`
			` * * *`
			`* * * *`
			` * * *`
			`  * *`
			`   *`

Video: Rhombus of Stars

Watch this video lesson to learn how to print a rhombus of stars on the console using nested loops: https://youtu.be/BaSgBU6yLU8.

Hints and Guidelines

To solve this problem, we need to mentally divide the rhombus into two parts – upper one, which also includes the middle row, and lower one. For the printing of each part we will use two separate loops, as we leave the reader to decide the dependency between **n** and the variables of the loops. For the first loop we can use the following guidelines:

- We print **n-row** white spaces.
- We print *****.
- We print **row-1** times *****.

The second (lower) part will be printed similarly, which again we leave to the reader to do.

```
for (var row = 1; row <= n; row++)
{
    for (var col = 1; col <= n - row; col++)
    {
        Console.Write(" ");
    }
    Console.Write("*");
    for (var col = 1; col < row; col++)
    {
        Console.Write(" *");
    }
    Console.WriteLine();
}
// TODO: print the down side of the rhombus
```

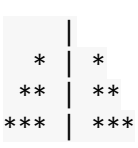
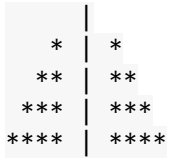
Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/512#5.

Problem: Christmas Tree

Write a program that takes a number **n** (1 ≤ n ≤ 100) and prints a Christmas tree with height of n+1.

Input	Output	Input	Output
1		2	

Input	Output	Input	Output
3		4	

Video: Christmas Tree

Watch this video lesson to learn how to print a Christmas tree on the console using nested loops: https://youtu.be/UecoBfhUlkk.

Hints and Guidelines

From the examples we see that the Christmas tree can be divided into three logical parts. The first part is the stars and the white spaces before and after them, the middle part is **|**, and the last part is again stars, but this time there are white spaces only before them. The printing can be done with only one loop and the **`new string(…)`** constructor, which we will use once for the stars and once for the white spaces.

```
int n = int.Parse(Console.ReadLine());
for (int i = 0; i <= n; i++)
{
    var stars = new string('*', i);
    var spaces = new string(' ', n - i);
    Console.Write(spaces);
    Console.Write(stars);
    Console.Write(" | ");
    Console.Write(stars);
    Console.WriteLine(spaces);
}
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/512#6.

Problem: Sunglasses

Write a program that takes an integer **n** (3 ≤ n ≤ 100) and prints sunglasses with size of **5*n x n** as found in the examples:

Input	Output	Input	Output
3		4	

Input	Output
5	********** ********** */////////* */////////* */////////*|||||*/////////* */////////* */////////* ********** **********

Video: Sunglasses

Watch this video lesson to learn how to print sunglasses on the console using nested loops: https://youtu.be/MTQhldgno4k.

Printing the Top and Bottom Rows

From the examples we can see that the sunglasses can be divided into three parts – upper, middle and lower one. A part of the code with which the problem can be solved is given below.

When drawing the upper and lower rows we need to print `2 * n` stars, `n` white spaces and `2 * n` stars.

```
// Print the top part
Console.Write(new string('*', 2 * n));
Console.Write(new string(' ', n));
Console.WriteLine(new string('*', 2 * n));
for (int i = 0; i < n - 2; i++)
{
    // TODO: print the middle part
}
// Print the bottom part
Console.Write(new string('*', 2 * n));
Console.Write(new string(' ', n));
Console.WriteLine(new string('*', 2 * n));
```

Printing the Middle Rows

When drawing the middle part, we need to check if the row is `(n-1) / 2 - 1`, because in the examples we can see that in this row we need to print pipes instead of white spaces.

```
// Print the middle part
for (int i = 0; i < n - 2; i++)
{
    // TODO: print */////*
    if (i == (n - 1) / 2 - 1)
        Console.Write(new string('|', n));
```

```
    else
        Console.Write(new string(' ', n));
    // TODO: print *///////*
    Console.WriteLine();
}
```

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/512#7.

Problem: House

Write a program that takes a number **n** ($2 \leq n \leq 100$) and prints **a house** with size **n x n**, just as in the examples:

Input	Output	Input	Output	Input	Output
2	** ||	3	-*- *** |*|	4	-**- **** |**| |**|

Input	Output	Input	Output
5	--*-- -***- ***** |***| |***|	8	---**--- --****-- -******- ******** |******| |******| |******| |******|

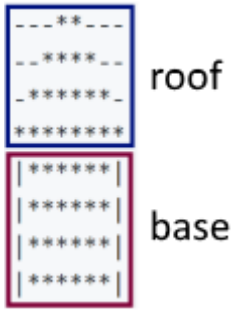
Video: Draw a House

Watch this video lesson to learn how to print a house on the console using nested loops: https://youtu.be/ExjxRM0vXW4.

Hints and Guidelines

We understand from the problem explanation that the house is with size of **n** x **n**. What we see from the example input and output is that:

- The house is divided into two parts: **roof and base**.
- When **n** is an even number, the point of the house is "dull".
- When **n** is odd, **the roof** is one row larger than the **base**.



Roof

- It comprises of **stars** and **dashes**.

- In the top part there are one or two stars, depending on if **n** is even or odd (also related to the dashes).
- In the lowest part there are many stars and no dashes.
- With each lower row, the stars increase by 2 and the dashes decrease by 2.

Base

- The height is **n** rows.
- It is made out of stars and pipes.
- Each row comprises of 2 pipes – one in the beginning and one in the end of the row, and also stars between the pipes with string length of **n - 2**.

Reading the Input Data

We read **n** from the console and we save it in a variable of **int** type.

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

It is very important to check if the input data is correct! In these tasks it is not a problem to directly convert the data from the console into **int** type, because it is said that we will be given valid integers. If you are making more complex programs it is a good practice to check the data. What will happen if instead of the character "A" the user inputs a number?

Calculating Roof Length

In order to draw the roof, we write down how many stars we start with in a variable called **stars**:

- If **n** is an even number, there will be 2 stars.
- If it is odd, there will be 1.

```
var stars = 1;
if (n % 2 == 0)
{
    stars++;
}
```

Calculate the length of the roof. It equals half of **n**. Write the result in the variable **roofLength**.

```
var roofLength = (int)Math.Ceiling(n / 2f);
```

It is important to note that when **n** is an odd number, the length of the roof is one row more than that of the base. In C# when you divide two numbers with a remainder, the result will be the number without remainder.

Example:

```
int result = 3 / 2; // result 1
```

If we want to round up, we need to use the method **Math.Ceiling(…)**: **int result = (int)Math.Ceiling(3 / 2f);** In this example the division isn't between two integers. "**f**"

after a number shows that this number is of **float** type (a floating-point number). The result of **3 / 2f** is **1.5f**. **Math.Ceiling(…)** rounds the division up. In this case **1.5f** will become 2. **(int)** is used so that we can transfer the type back to **int**.

Printing the Roof

After we have calculated the length of the roof, we make a loop from 0 to **roofLength**. On each iteration we will:

- Calculate the number of dashes we need to draw. The number will be equal to **(n - stars) / 2**. We store it in a variable **padding**.

```
var padding = (n - stars) / 2;
```

- We print on the console: "dashes" (**padding / 2** times) + "stars" (**stars** times) + "dashes" (**padding / 2** times).

```
var line = new string('-', padding)
    + new string('*', stars)
    + new string('-', padding);
Console.WriteLine(line);
```

- Before the iteration is over, we add 2 to **stars** (the number of the stars).

```
stars += 2;
```

It is not a good idea to add many character strings as it is shown above, because this leads to performance issues. Learn more at: https://en.wikipedia.org/wiki/String_(computer_science)#String_buffers

Printing the Base

After we have finished with the roof, it is time for the base. It is easier to print:

- We start with a loop from 0 to n (not inclusive).
- We print on the console: **|** + ***** (**n - 2** times) + **|**.

```
for (int i = 0; i < n / 2; i++)
{
    var line = "|" + new string('*', n - 2) + "|";
    Console.WriteLine(line);
}
```

If you have written everything as it is here, the problem should be solved.

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/512#8.

Problem: Diamond

Write a program that takes an integer **n** ($1 \le n \le 100$) and prints a diamond with size **n**, as in the following examples:

Input	Output	Input	Output	Input	Output

1	`*`	2	`**`	3	`-*-`
					`*-*`
					`-*-`

Input	Output	Input	Output	Input	Output

4	`-**-`
	`*--*`
	`-**-`
		5	`--*--`
			`-*-*-`
			`*---*`
			`-*-*-`
			`--*--`
				6	`--**--`
					`-*--*-`
					`*----*`
					`-*--*-`
					`--**--`

Input	Output	Input	Output	Input	Output

7	`---*---`
	`--*-*--`
	`-*---*-`
	`*-----*`
	`-*---*-`
	`--*-*--`
	`---*---`
		8	`---**---`
			`--*--*--`
			`-*----*-`
			`*------*`
			`-*----*-`
			`--*--*--`
			`---**---`
				9	`----*----`
					`---*-*---`
					`--*---*--`
					`-*-----*-`
					`*-------*`
					`-*-----*-`
					`--*---*--`
					`---*-*---`
					`----*----`

Video: Draw a Diamond

Watch this video lesson to learn how to print a diamond on the console using nested loops: https://youtu.be/Z8crtxDztBk.

Hints and Guidelines

What we know from the problem explanation is that the diamond is with size **n** x **n**.

From the example input and output we can conclude that all rows contain exactly **n** symbols, and all the rows, with the exception of the top and bottom ones, have 2 stars. We can mentally divide the diamond into 2 parts:

- Upper part. It starts from the upper tip down to the middle.
- Lower part. It starts from the row below the middle one and goes down to the lower tip (inclusive).

Upper Part

- If n is an odd number, it starts with 1 star.
- If n is an even number, it starts with 2 stars.
- With each row down, the stars get further away from each other.
- The space between, before and after the stars is filled up with dashes.

Lower Part

- With each row down, the stars get closer to each other. This means that the space (the dashes) between them is getting smaller and the space (the dashes) on the left and on the right is getting larger.
- The bottom-most part has 1 or 2 stars, depending on whether n is an even or odd number.

Upper and Lower Parts of the Diamond

- On each row, except the middle one, the stars are surrounded by inner and outer dashes.
- On each row there is space between the two stars, except on the first and the last row (sometimes the star is 1).

Reading the Input Data

We read n from the console and we save it in a variable of `int` type.

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

Printing the Top Part of the Diamond

We start drawing the upper part of the diamond. The first thing we need to do is to calculate the number of the outer dashes `leftRight` (the dashes on the outer side of the stars). It is equal to `(n - 1) / 2`, rounded down.

```
var leftRight = (n - 1) / 2;
```

After we have calculated `leftRight`, we start drawing the upper part of the diamond. We can start by running a loop from `0` to `n / 2 + 1`(rounded down).

At each iteration of the loop the following steps must be taken:

- We draw on the console the left dashes (with length `leftRight`) and right after them the first star.

```
Console.WriteLine(new string('-', leftRight));
Console.Write("*");
```

- We will calculate the distance between the two stars. We can do this by subtracting from n the number of the outer dashes, and the number 2 (the number of the stars, i.e. the diamonds outline). We need to store the result of the subtraction in a variable `mid`.

```
var mid = n - 2 * leftRight - 2;
```

- If `mid` is lower than 0, we know that on the row there should be only 1 star. If it is higher or equal to 0 then we have to print dashes with length `mid` and one star after them.
- We draw on the console the right outer dashes with length `leftRight`.

```
Console.WriteLine(new string('-', leftRight));
```

- In the end of the loop we decrease `leftRight` by 1 (the stars are moving away from each other).

We are ready with the upper part.

Printing the Bottom Part of the Diamond

Printing the lower part is very similar to that of the upper part. The difference is that instead of decreasing **`leftRight`** with 1 in the end of the loop, we will increase **`leftRight`** with 1 at the beginning of the loop. Also, the loop will be from 0 to **`(n - 1) / 2`**.

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

Repeating a code is considered bad practice, because the code becomes very hard to maintain. Let's imagine that we have a piece of code (e.g. the logic for drawing a row from the diamond) at a few more places and we decide to change it. For this we will have to go through all the places and change it everywhere. Now let's imagine that you need to reuse a piece of code not 1, 2 or 3 times but tens of times. A way to overcome this problem is to use methods.

If we have written all correctly, then the problem is solved.

Testing in the Judge System

Test your solution here: https://judge.softuni.org/Contests/Practice/Index/512#9.

Lab: Drawing Ratings in Web

Now that we got used to nested loops and the way to use them to draw figures on the console, we can get into something even more interesting: we can see how loops can be used to draw in a Web environment. We will make a web application that visualizes a number rating (a number from 0 to 100) with stars. This kind of visualization is common in e-commerce sites, reviews of products, event rating, rating of apps, and others.

Don't worry if you don't understand all of the code, how exactly it is written and how the project works. It is normal, now we are learning to write code and we are a long way from the web development technologies.

Video: Building a Web App "Draw Ratings"

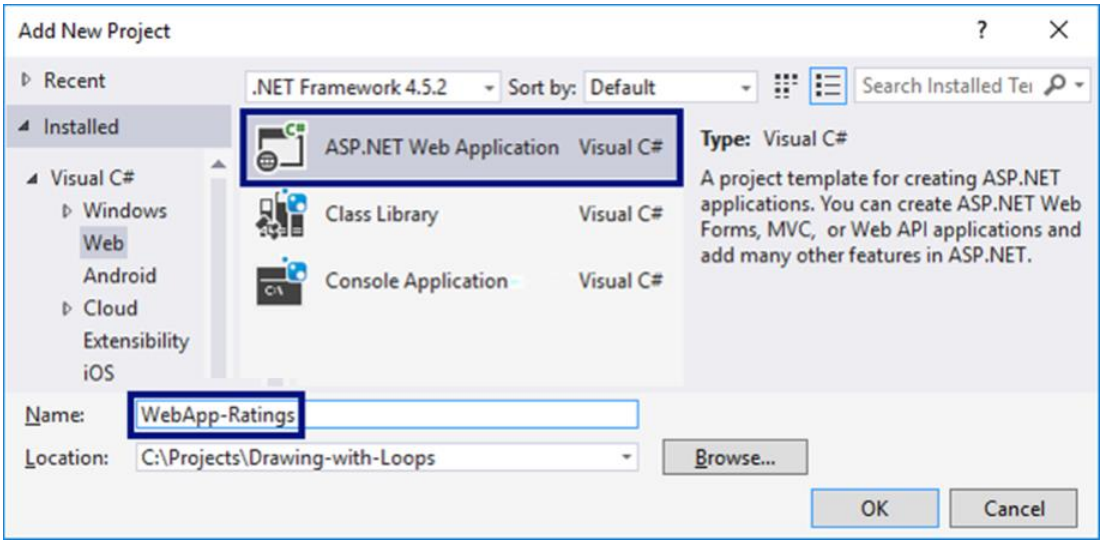
Watch a video lesson to learn how to build ASP.NET MVC Web app to draw ratings by given number: https://youtu.be/ErnapuBJvZQ.

Ratings – Visualization in a Web Environment

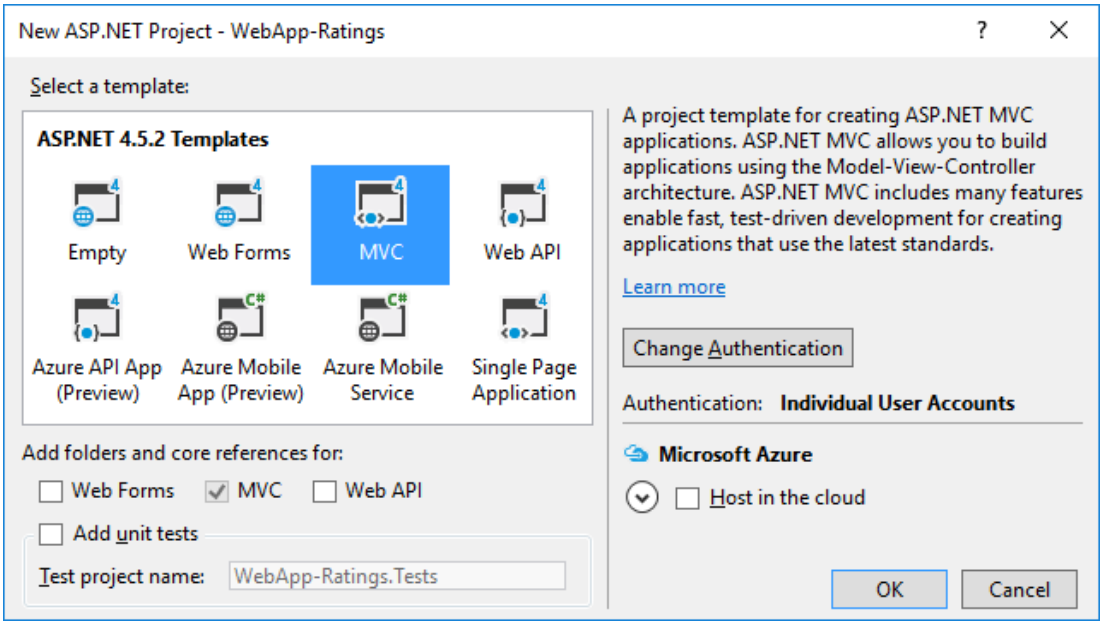
Your task now is to create an ASP.NET MVC Web Application for visualizing a rating (a number from 0 to 100). 1 to 10 stars should be drawn (with halves). The stars should be generated with a **`for`** loop.

Creating a New C# Project

Create a new ASP.NET MVC web app with C# in Visual Studio. Add a new project from [Solution Explorer] -> [Add] -> [New Project...]. Give it a meaningful name, for example "WebApp-Ratings".

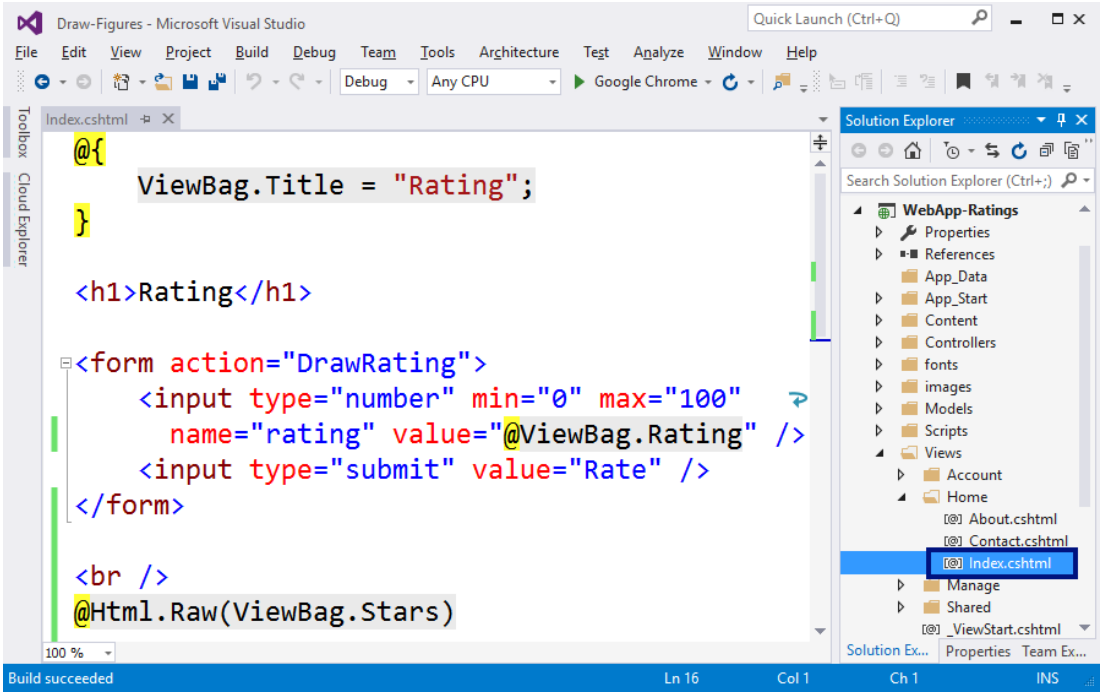


Choose the type of the app to be MVC.



Creating a View Holding a HTML Form

Open and edit the file `Views/Home/Index.cshtml`. Delete everything and insert the following code:



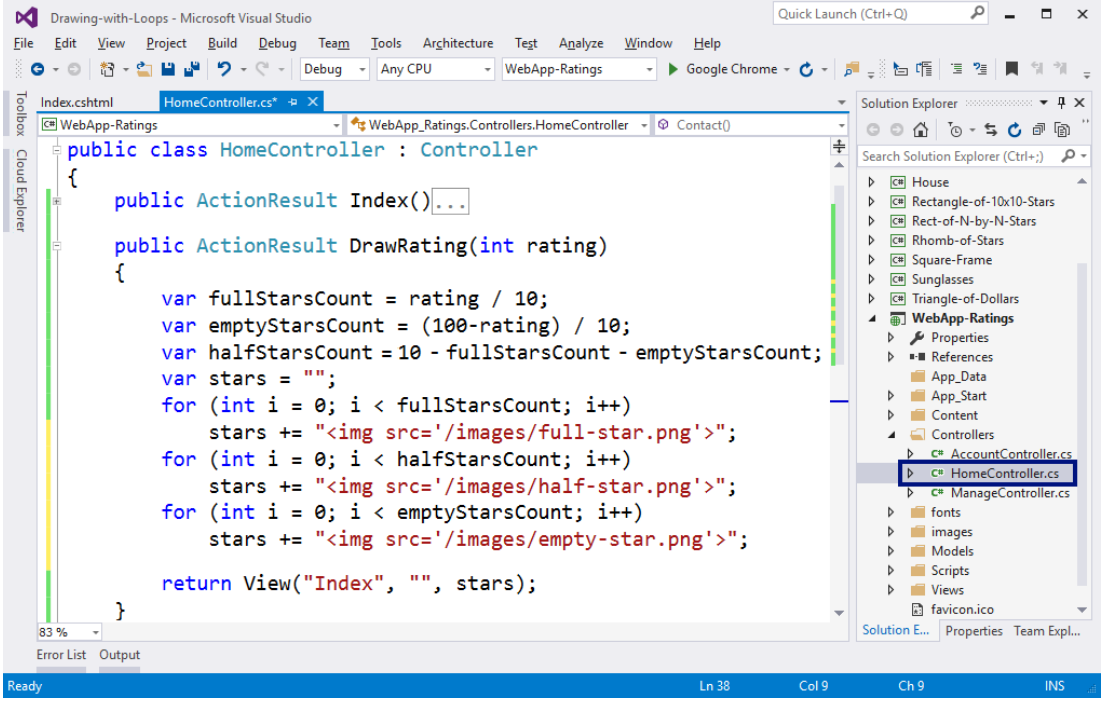
This code creates a web form `<form>` with a field `"rating"` for inputting a number in the range [0 ... 100] and a button [Draw] to send data from the form to the server. The action that will process the data is called `/Home/DrawRatings`, which means method `DrawRatings` in controller `Home`, which is in the file `HomeController.cs`. After the form the contents of `ViewBag.Stars` are printed. The code that will be inside will be dynamically generated by the HTML controller with a series of stars.

Adding the DrawRatings(int) Method

Add a method `DrawRatings` in the controller `HomeController`. Open the file `Controllers/HomeController.cs` and add the following code:

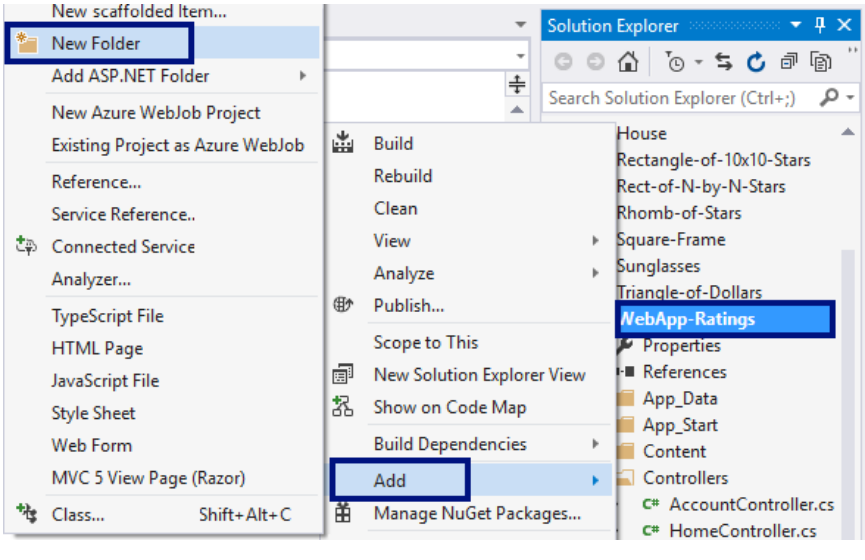
```
public ActionResult DrawRating(int rating)
{
    var fullStars = rating * 10 / 100;
    var emptyStars = (100 - rating) * 10 / 100;
    var halfStars = 10 - fullStars - emptyStars;

    var stars = "";
    for (int i = 0; i < fullStars; i++)
        stars += "<img src='/images/full-star.png' />";
    for (int i = 0; i < halfStars; i++)
        stars += "<img src='/images/half-star.png' />";
    for (int i = 0; i < emptyStars; i++)
        stars += "<img src='/images/empty-star.png' />";

    ViewBag.Stars = stars;
    ViewBag.Rating = rating;
    return View("Index");
}
```

The above code takes the number `rating`, makes some calculations to find the number of **full stars**, the number of **empty stars** and the number of **half-full stars**, after which it generates an HTML code, which orders a few pictures of stars one after the other so that it can make the rating picture from them. The ready HTML code is stored in `ViewBag.Stars` to visualize the view `Index.cshtml`. Additionally, the sent rating is kept (as a number) in `ViewBag.Rating`, so that it can be put in the field for rating in the view. In order to ease your work, you can help yourself with the picture of Visual Studio below:

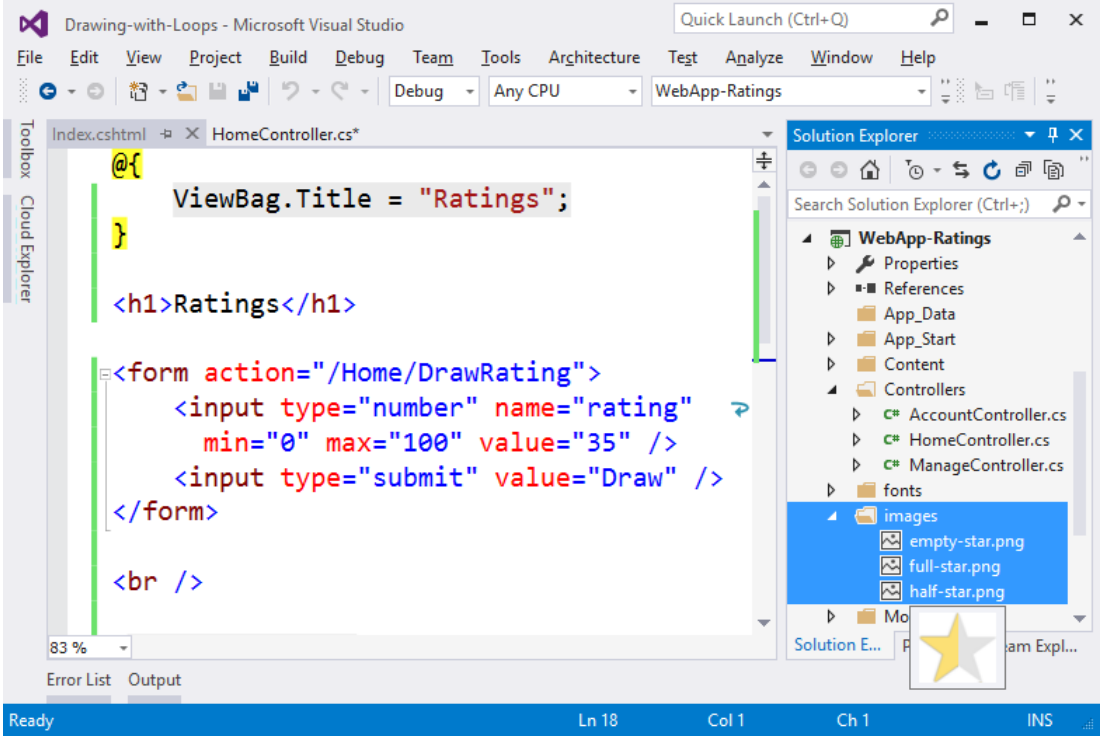


Adding Star Images

Create a new **images** folder in the project, using [**Solution Explorer**]:

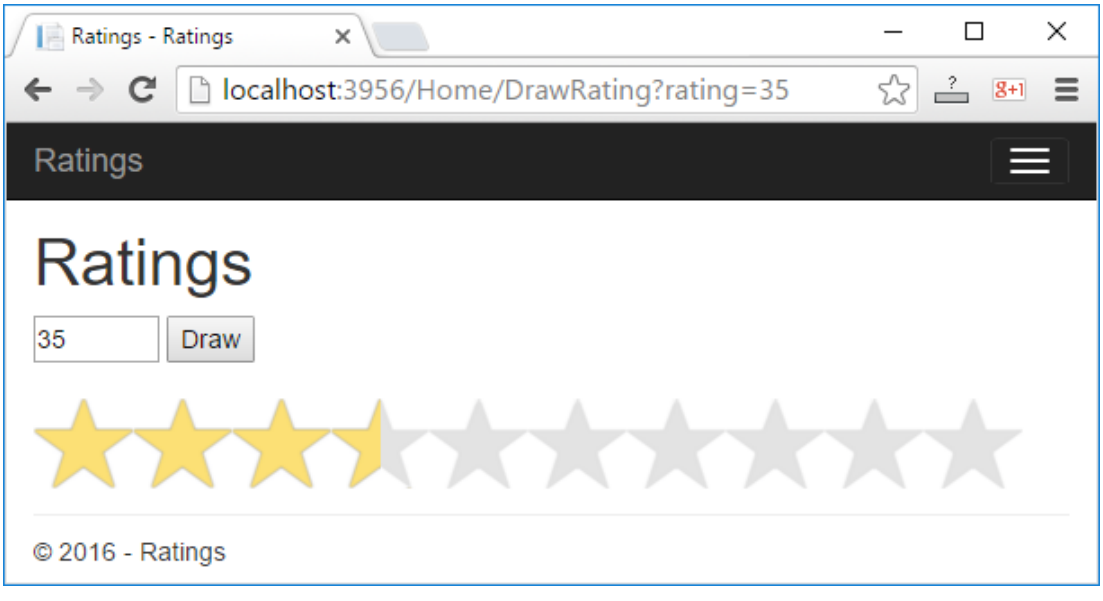


Now add the pictures with the stars (they are a part of the files with this project and can be downloaded from the book's GitHub repository: https://github.com/SoftUni/Programming-Basics-Book-CSharp-EN/tree/master/assets/chapter-6-assets). Copy them from Windows Explorer and paste them in the images folder in [Solution Explorer] in Visual Studio.



Starting and Testing the Project

Start the project with [Ctrl+F5] and enjoy:



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!