MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
NATIONAL TECHNICAL UNIVERSITY
“KHARKIV POLYTECHNIC INSTITUTE”

O.P. Arsenyeva, L.V. Solovey

C# language programming

Theoretical and practical guide
in «Computer Science»
(in English)

for the students of oil and gas specialization
In two volumes
Volume 1

APPROVED
by the publishing council of the university,
record № 2 from 17.05.2019 p.

Kharkiv
NTU «KhPI»
2019

УДК 004.43(075)
А85

Reviewers:

М.Л. Угрюмов, д.т.н., проф., Харківський національний університет імені В.Н. Каразіна МОН України, професор кафедри теоретичної та прикладної системотехніки

О.І. Горошко, завідуюча кафедри міжкультурної комунікації та іноземної мови, проф., д.с.н., д.ф.н.

Навчально-методичний посібник присвячений вивченню мови програмування C# 4.5. Наведено велику кількість прикладів написання програм різної складності. До прикладів надаються пояснення. Усі програми забезпечені результатами виконання. До кожної теми подані практичні завдання для виконання лабораторних робіт. Матеріал розміщено за принципом поступовості, починаючи з основ до більш складних можливостей мови програмування C #.

Призначено для студентів спеціальності 161 «Хімічна технологія та інженерія».

Арсеньєва О. П.

А85 Програмування мовою C#: навч. метод. посіб. / О. П. Арсеньєва, Л. В. Соловей. – Харків: НТУ «ХПІ», 2019. – 104 с. – Англ. мовою.

The present theoretical and practical guide concerns the study of C# programming language, version 4.5. The guide includes a lot of examples of coding with different difficulty level. The examples contain the theoretical explanations of each coding part. All the mentioned programs are given with the outgoing results. Each subject contains the practical examples for individual work or laboratory work. The guide is structured according to the complexity approach, starting from the basic knowledge and step-by step explanation of more specific features of C# programming language.

The guide contains the education material for 161st specialty namely “Chemical technology and engineering”.

Іл.: 60 Табл.: 13 Бібліогр.:

УДК 004.43(075)

Content

Introduction

There exist many programming languages, but only a few of them are really good. A good programming language should be efficient and flexible simultaneously, and its syntax should be brief, but clear. It should support the most advanced features of programming. The programming language C # is exactly such a language.

C # was created by Microsoft to support the .NET Framework environment and is based on considerable achievements in the field of coding. Its main developer was Anders Heilsberg (Anders Hejlsberg) – a well-known specialist in programming. C # comes directly from the two most successful programming languages in the field, namely: C and C ++. It inherited the syntax, many keywords and operators from C, and the advanced object model from C ++. In addition, C # is closely related to Java – no less than other good languages.

The present practical guide contains both the brief information about the approaches and methods for the coding in C # and examples of practical coding with suggested tasks for the individual work.

Volume 1 of the guide consists of four separate Chapters. In Chapter 1 the methods for the simple programs with linear structure are provided. Chapter discusses the programs with branch structure. The programs with iterations are listed in Chapter 3, and coding using the vectors and one-dimensional arrays is described in Chapter 5. Each chapter contains the explanation and coding example of creating the programs for console application without using object-oriented programming, and for Windows application it is explained how to create the user-defined forms for input and output information.

Chapter 1. Programs with linear structure

Objectives:

1. For console applications
   To study the methods of initializing and assignment of variables and constants; methods of declaration, input and output, creating comments in programming code – Console.Write(), Console.WriteLine(), Console.Read(), Console.ReadLine().
2. For Windows applications
   To study the Common Controls elements: Label, Button, TextBox.

Task 1.1. Calculate the value of the expression.

$$y = btg^2 x - \frac{a}{\sin^2(x/a)};$$

$$d = ae^{-\sqrt{a}}\cos(bx/a);$$

where $a = 3.2;\ b = 17.5;\ x = -4.8.$

Solution 1.

In this Solution the program is written as the console application with the data input as constants.

Console applications are the simplest form of C# program, which perform all their input and output at the command line, they are ideal for quick understanding the language features and writing command-line utilities.

Solution 1 enters the initial data as constants in the programming code, and the results are outputed to the command line. The flowchart of the solution1 is presented in Fig. 1.1.

To create a C# console application

1. Open Microsoft Visual Studio 2012.
Пуск /Программы /Программирование /Microsoft Visual Studio 2012 / ∞ Microsoft Visual Studio 2012
The start window of the program appears (Fig. 1.2).

2. On the **File** menu, click **New Project**.
The **New Project** dialog box appears. This dialog box lists the different default application types that Visual C# Express Edition can create (Fig. 1.3).

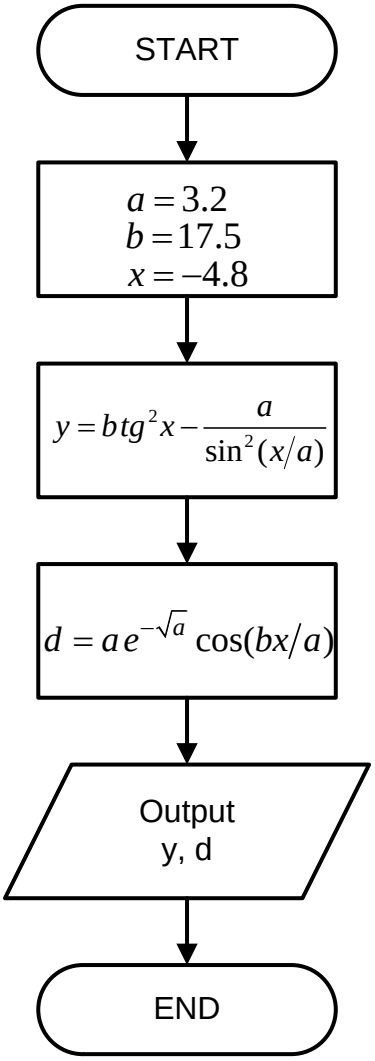


Figure 1.1 - Flowchart of solution 1

3. Select **Console Application** as your project type and change the name of your application to ***Solution1***.
The application environment should be **.NET Framework 4**.
Language (left column): **C#**.
Select **Console Application** visual C# to create new console application.

The default name and location should be defined, but you can always enter a new path if you want. The default name is *ConsoleApplication1*, you need to change it for ***Solution1***, and the path: ***D:/Documents/O65/Programs***.

4. Click **OK**.

Visual Studio 2012creates a new folder for your project named after the project title. It also opens the main Visual C# Express Edition window and the Code pane where you will enter and modify the C# source code that creates your application (Fig. 1.4).

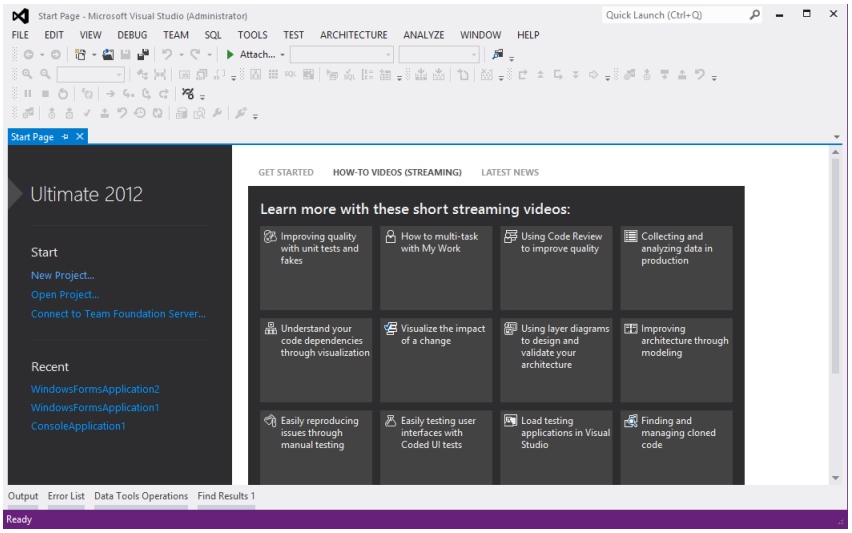


Figure 1.2 - Initial window of Microsoft Visual Studio 2012

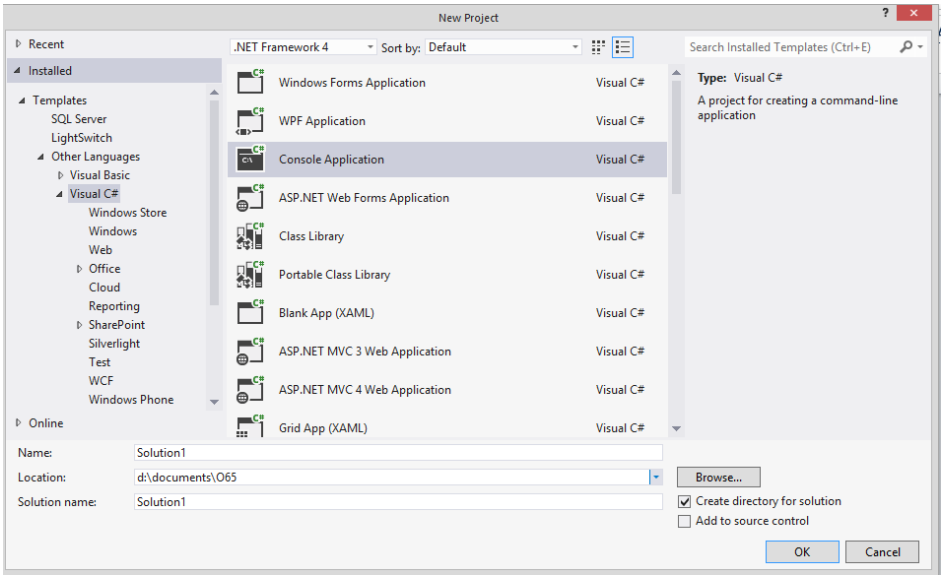


Figure 1.3 - **New project** dialog box

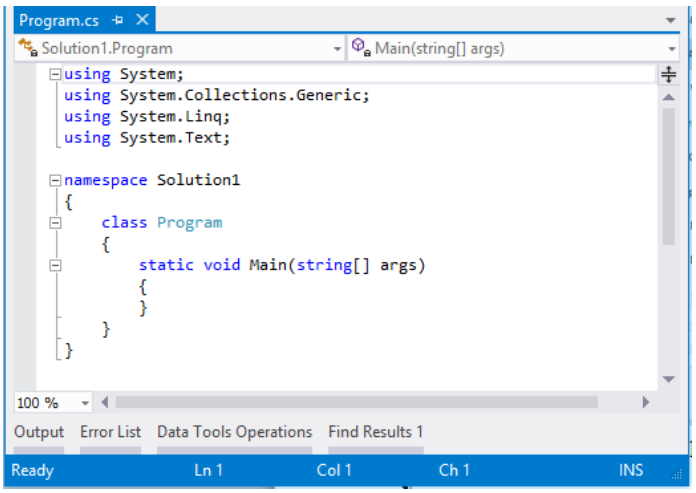


Figure 1.4 - Visual C# code pane

5. Enter the following C# source code for solving the Task 1.1:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Solution1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Program 1
            const double a = 3.2, b = 17.5, x = -4.8;
            double y, d;
            y = b * Math.Pow(Math.Tan(x), 2) - a /
                          Math.Pow(Math.Sin(x / a), 2);
            d = a * Math.Exp(-Math.Sqrt(a)) * Math.Cos(b*x/a);
            Console.WriteLine("y={0,9:F3};  d={1,8:F4}", y, d);
            Console.WriteLine("Press Enter");
            Console.ReadLine(); /* You need ReadLine operator
            to stop to see the command line */
        }
    }
}
```

6. Remove Unused Usings from the first part of the code (Fig. 1.5).

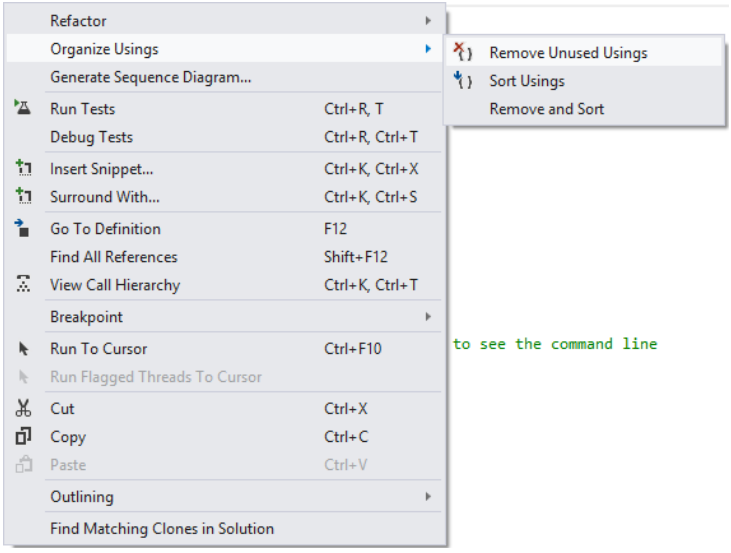


Figure 1.5 - Remove the usings, which are not used

The new C# source code:

```
using System;
namespace Solution1
{
    class Program
    {
        static void Main(string[] args)
        {
            // Program 1
            const double a = 3.2, b = 17.5, x = -4.8;
            double y, d;
            y = b * Math.Pow(Math.Tan(x), 2) - a /
                                Math.Pow(Math.Sin(x / a), 2);
            d = a * Math.Exp(-Math.Sqrt(a)) * Math.Cos(b*x/a);
            Console.WriteLine("y={0,9:F3};  d={1,8:F4}", y, d);
            Console.WriteLine("Press Enter");
            Console.ReadLine(); /* You need ReadLine operator
            to stop to see the command line */
        }
    }
}
```

7. Run the program.
Press Start button (Fig. 1.6) or F5.

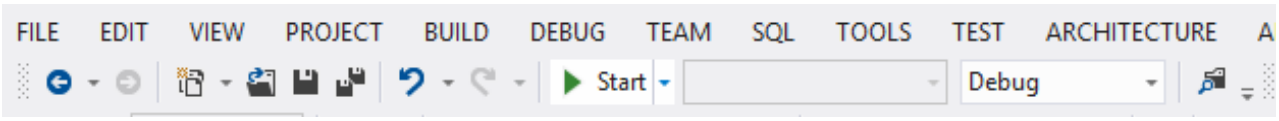


Figure 1.6 - Start button to run the program

It will give the results of calculations displayed in command line (Fig. 1.7).

8. Save your Project (Fig. 1.8).
System will save your project: *D:\ Documents \O65\Programs\Solution1\ Solution1.sln*.

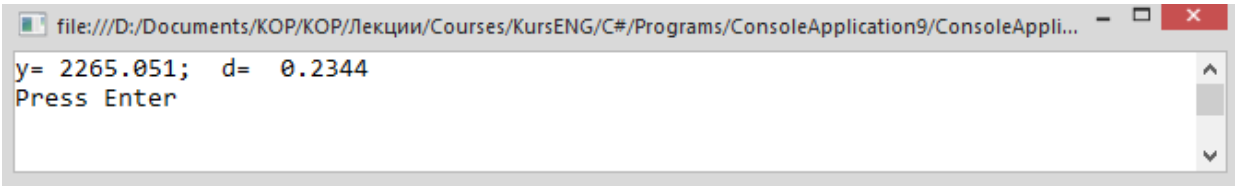


Figure 1.7 - Command line with the results of calculations

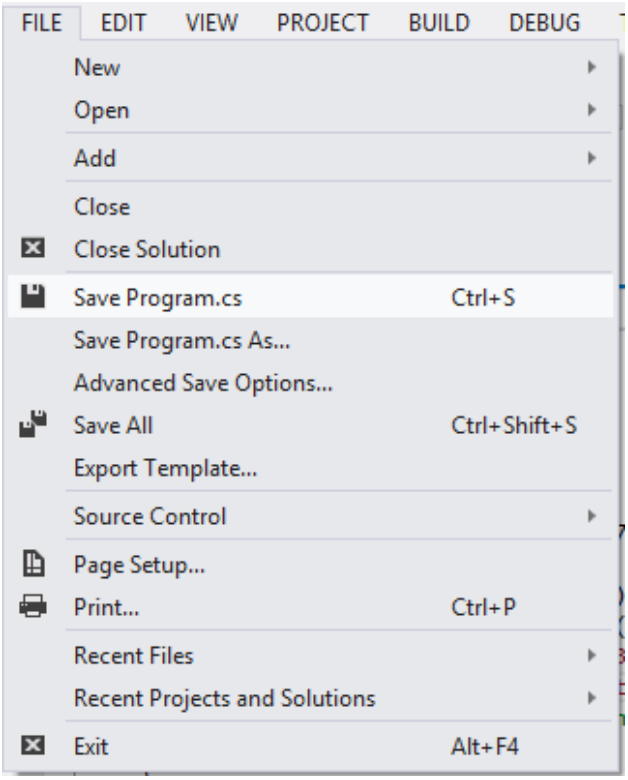


Figure 1.8 - **File** menu

The explanation of the C# code, which is used in the example (Solution 1):

The key organizational concepts in C# are ***programs***, ***namespaces***, ***types***, ***members***, and ***assemblies***. C# programs consist of one or more source files. Programs declare types, which contain members and can be organized into namespaces. Classes and interfaces are examples of types. Fields, methods, properties, and events are examples of members. When C# programs are compiled, they are physically packaged into assemblies. Assemblies typically have the file extension .exe or .dll, depending on whether they implement ***applications*** or ***libraries***.

1) `using System`

Most C# applications begin with a section of **using** directives. This section lists the *namespaces* that the created application will use frequently, and helps the

programmer to specify a full name with all descriptions each time when a called method is used within a program. *Namespaces* come in two categories: user-defined and system-defined. *User-defined namespaces* are the *namespaces*, which are defined in your code. For a list of system-defined namespaces, see .NET Framework Class Library.

For example, include the line:

```
using System;
```

At the start of a program, the programmer can use the code:

```
Console.WriteLine("Hello, World!");
```

Instead of:

```
System.Console.WriteLine("Hello, World!").
```

2) `namespace Solution1`

When you create a program in Visual C# Express, a *namespace* is automatically created for you. To use classes from other namespaces in your program, you must specify them with a **using Directive**. The most commonly used.NET Framework namespaces are listed by default when you create a new application. In this case it takes the name of file: Solution1. If you use classes from other namespaces in the class library, you must add a using directive for that namespace to the source file.

3) `class Program`

A class is a construct that enables you to create your own custom types by grouping together variables of other types, methods and events. A class is like a blueprint. It defines the data and behavior of a type. If the class is not declared as static, client code can use it by creating objects or instances which are assigned to a variable. The variable remains in memory until all references to it go out of scope. If the class is declared as static, then only one copy exists in memory and client code can only access it through the class itself, not an instance variable.

4) `static void Main()`

The **Main** method is the entry point of a C# console application or windows application. (Libraries and services do not require a **Main** method as an entry point). When the application is started, the **Main** method is the first method that is invoked.

There can be only one entry point in a C# program. If you have more than one class that has a **Main** method, you must compile your program with the **/main** compiler option to specify which **Main** method to use as the entry point.

- The **Main** method is the entry point of an .exe program; it is where the program control starts and ends.
- **Main** is declared inside a class or struct. **Main** must be *static* and it should not be *public*. The enclosing class or struct is not required to be static.
- **Main** can either have a **void** or **int** return type. `Void` means that method **Main** does not return the value.
- The **Main** method can be declared with or without a string[] parameter that contains command-line arguments. When using Visual Studio to create Windows Forms applications, you can add the parameter manually or else use the *Environment* class to obtain the command-line arguments. Parameters are read as zero-indexed command-line arguments.

This line indicates the starting of the C# source code. The *block* of code for the method starts and ends with the *Braces* (`{ }`). This code is executed when you run your console application. Well, to be more precise, the code block enclosed in curly braces is executed.

5) **Statements**

The actions of a program are expressed using ***statements***. C# code is made up of a series of statements, each of which is terminated with a **semicolon**. Because whitespace is ignored, multiple statements can appear on one line, although for readability it is usual to add carriage returns after semicolons, to avoid multiple statements on one line. It is perfectly acceptable (and quite normal), however, to use statements that span several lines of code.

C# is a **block**-structured language, meaning statements are part of a **block** of code. These **blocks**, which are delimited with curly brackets (`{` and `}`), may contain any number of statements, or none at all. *Note that the curly bracket characters do not need accompanying semicolons*.

C# supports several different kinds of statements, a number of which are defined in terms of embedded statements.

- A ***block*** permits multiple statements to be written in contexts where a single statement is allowed. A block consists of a list of statements written between the delimiters { and }.

- ***Declaration statements*** are used to declare local variables and constants.
- ***Expression statements*** are used to evaluate expressions. Expressions that can be used as statements include method invocations, object allocations using the new operator, assignments using = and the compound assignment operators, increment and decrement operations using the ++ and -- operators and await expressions.
- ***Selection statements*** are used to select one number from possible statements for execution based on the value of some expression. In this group if and switch statements exist.
- ***Iteration statements*** are used to repeatedly execute an embedded statement. The while, do, for, and foreach statements belong to this group.
- ***Jump statements*** are used to transfer control. The break, continue, goto, throw, return, and yield statements are included to this group.
- The checked and unchecked statements are used to control the overflow checking context for integral-type arithmetic operations and conversions.
- The lock statement is used to obtain the mutual-exclusion lock for a given object, execute a statement, and then release the lock.
- The using statement is used to obtain a resource, execute a statement, and then dispose of that resource.

Declarations in a C# program define the constituent elements of the program. C# programs are organized using namespaces, which can contain type declarations and nested namespace declarations. A declaration defines a name in the ***declaration space*** to which the declaration belongs.

6) `// Program 1`

It is indication of comments in C#. A comment is not, strictly speaking, C# code at all, but it happily cohabits with it. Comments are self-explanatory: they enable you to add descriptive text to your code—in plain English (or French, German, Mongolian, and so on) - which is *ignored by the compiler*.

C# provides two ways of doing this. You can either place markers at the beginning and end of a comment or you can use a marker that means ″everything on the rest of this line is a comment.″

The latter method is an exception to the rule mentioned previously about C# compilers ignoring carriage returns, but it is a special case.

To indicate comments using the first method, you use **`/*`** characters at the start of the comment and **`*/`** characters at the end. These may occur on a single line,

or on different lines, in which case all lines in between are part of the comment. The only thing you can't type in the body of a comment is **`*/`**, because that is interpreted as the end marker.

The other commenting approach involves starting a comment with **`//`**. After that, you can write whatever you like - as long as you keep to one line.

```
7) const double a = 3.2, b = 17.5, x = -4.8;
```

You use the **const** keyword to declare a constant field or a constant local. Constant fields and locals *aren't variables* and *may not be modified*. Constants can be numbers, Boolean values, strings, or a null reference. Don't create a constant to represent information that you expect to change at any time.

Here the constants `a, b, x` are **double** type. The **double** keyword signifies a simple type that stores 64-bit floating-point values. By default, a real numeric literal on the right side of the assignment operator is treated as **double**.

```
8) double y, d;
```

Variables represent storage locations. Every variable has a type that determines what values can be stored in the variable. C# is a type-safe language, and the C# compiler guarantees that values stored in variables are always of the appropriate type. The value of a variable can be changed through assignment or through use of the ++ and -- operators.

The *type* of a *local-variable-declaration* specifies the type of the variables introduced by the declaration. The type is followed by a list of *local-variable-declarator*s, each of which introduces a new variable. A *local-variable-declarator* consists of an *identifier* that names the variable, optionally followed by an "=" token and a *local-variable-initializer* that gives the initial value of the variable.

The value of a local variable is obtained in an expression using a *simple-name*, and the value of a local variable is modified using an *assignment*. A local variable must be definitely assigned at each location where its value is obtained.

Here the variables `y, d` are declared and are **double** type. The values of these variables can be modified in the program using an *assignment*.

```
9) y = b * Math.Pow(Math.Tan(x), 2) - a/Math.Pow(Math.Sin(x/a), 2);
```

Expressions are constructed from ***operands*** and ***operators***. The operators of an expression indicate which operations to apply to the operands. Examples of

operators include +, -, *, /, and new. Examples of operands include literals, fields, local variables, and expressions.

When an expression contains multiple operators, the ***precedence*** of the operators controls the order in which the individual operators are evaluated. For example, the expression x + y * z is evaluated as x + (y * z) because the * operator has higher precedence than the + operator. The list of simple operators used in (9) is presented in Table 1.1.

Table 1.1. - The list of selected C# operators

Category	**Expression**	**Description**
Assignment	x = y	Assignment
	x+=y	The addition assignment operator meaning that x=x+y
	x-=y x*=y x/=y	The subtraction, multiplication and division assignment operator meaning that x=x-y; x=x*y and x=x/y
	x %= y	The % operator computes the remainder after division x/y
Multiplicative	x * y	Multiplication
	x / y	Division
	x % y	Remainder
Unary	x + y	Addition, string concatenation, delegate combination
	x – y	Subtraction, delegate removal
	!x	The logical negation operator.
Equality	==	Returns true if the values of its operands are equal, false otherwise
	!=	Returns false if its operands are equal, true otherwise
Primary	x.y	The dot operator is used for member access. It specifies a member of a type or namespace.
	a[x]	Square brackets ([]) are used for arrays, indexers, and attributes
	x++ ++x	The increment operator increments its operand by 1.
	x-- --x	The decrement operator decrements its operand by 1

Table 1.1 (continuation)

Category	Expression	Description
Relational and type testing	<	Returns true if the first operand is less than the second, false otherwise.
	>	Returns true if the first operand is greater than the second, false otherwise.
	<=	Returns true if the first operand is less than or equal to the second, false otherwise.
	>=	Returns true if the first operand is greater than or equal to the second, false otherwise.
Logical AND	&	For bool operands it computes the logical AND of its operands.
Logical XOR	^	For bool operands it computes the logical exclusive-or of its operands; that is, the result is true if and only if exactly one of its operands is true.
Logical OR	|	It computes the logical OR of its operands; that is, the result is false if and only if both its operands are false.
Conditional AND	&&	It performs a logical-AND of its bool operands, but only evaluates its second operand if necessary.
Conditional OR	||	It performs a logical-OR of its bool operands. If the first operand evaluates to true, the second operand isn't evaluated. If the first operand evaluates to false, the second operator determines whether the OR expression as a whole evaluates to true or false.

Math Class provides static methods for common mathematical functions. It contains constants and static methods for trigonometric, logarithmic, and other common mathematical functions. A ***method*** is a member that implements a

computation or action that can be performed by an object or class. Methods have a (possibly empty) list of ***parameters***, which represent values or variable references passed to the method, and a ***return type***, which specifies the type of the value computed and returned by the method. Parameters are used to pass values or variable references to methods. The parameters of a method get their actual values from the ***arguments*** that are specified when the method is invoked. There are four kinds of parameters: value parameters, reference parameters, output parameters, and parameter arrays. The common `Math` methods are listed in Table 1.2.

Table 1.2 - Some Math Class methods

Name	**Description**
1	2
Abs(Double)	Returns the absolute value of a double-precision floating-point number.
Acos(Double)	Returns the angle whose cosine is the specified number.
Asin(Double)	Returns the angle whose sine is the specified number.
Atan(Double)	Returns the angle whose tangent is the specified number.
Ceiling(Double)	Returns the smallest integral value that is greater than or equal to the specified.
Cos(Double)	Returns the cosine of the specified angle.
Exp(Double)	Returns **e** raised to the specified power.
Floor(Double)	Returns the largest integer less than or equal to the specified.
IEEERemainder(Double, Double)	Returns the remainder resulting from the division of a specified number by another specified number.
Log(Double)	Returns the natural (base **e**) logarithm of a specified number.
Log10(Double)	Returns the base 10 logarithm of a specified number.
PI	Represents the ratio of the circumference of a circle to its diameter, specified by the constant, π
Pow(Double, Double)	Returns a specified number raised to the specified power.

Table 1.2 (continuation)

1	2
Round(Double)	Rounds a double-precision floating-point value to the nearest integral value.
Round(Double, Int32)	Rounds a double-precision floating-point value to a specified number of fractional digits.
Sign(Double)	Returns a value indicating the sign of a double-precision floating-point number: $Sign(x) = \begin{cases} -1 & x < 0 \\ 0 & \text{if } x = 0 \\ +1 & x > 0 \end{cases}$
Sin(Double)	Returns the sine of the specified angle.
Sqrt(Double)	Returns the square root of a specified number.
Tan(Double)	Returns the tangent of the specified angle.
Truncate(Double)	Calculates the integral part of a specified double-precision floating-point number.

The `Math` methods from the expression (9) are explained in Table 1.3.

Table 1.3 - `Math` methods from the expression (9)

Math expression	**C# code**
$tg^2 x$	`Math.Pow(Math.Tan(x), 2)`
	Takes method `Pow(Variable1, Variable2)` from `Math` class, where `Variable1, Variable2` are ***double*** type. It returns a `Variable1` raised to the `Variable2` power.
	`Math.Tan(x)` - takes method `Tan(Variable1)` from `Math` class, where `Variable1` is ***double*** type. It returns the tangent of the `Variable1` angle.
$\sin^2(x/a)$	`Math.Pow(Math.Sin(x / a), 2)`
	`Math.Sin(x / a)` - takes method `Sin(Variable1)` from `Math` class, where `Variable1` is ***double*** type. It returns the sine of the `Variable1` angle.

10) `Console.WriteLine("y={0,9:F3};  d={1,8:F4}", y, d);`

The output of the program is produced by the WriteLine method of the Console class in the System namespace. This class is provided by the .NET Framework class libraries, which, by default, are automatically referenced by the Microsoft C# compiler. Console Class represents the standard input, output, and error streams for console applications. You can format numeric results by using the `String.Format` method, or through the `Console.Write` or `Console.WriteLine` method, which calls **String.Format**. The format is specified by using format strings. Table 1.4 contains the supported standard format strings.

Table 1.4 - The supported standard format strings

Format Specifier	**Description**	**Examples**	**Output**
C or c	Currency	Console.Write("{0:C}", 2.5);	$2.50
D or d	Decimal	Console.Write("{0:D5}", 25);	00025
E or e	Scientific	Console.Write("{0:E}", 250000);	2.500000E+005
F or f	Fixed-point	Console.Write("{0:F2}", 25); Console.Write("{0:F0}", 25);	25.00 25
G or g	General	Console.Write("{0:G}", 2.5);	2.5

The syntax of a format item is **{***index*[**,***alignment*][**:***formatString*]**}**, which specifies a mandatory index, the optional length and alignment of the formatted text, and an optional string of format specifier characters that govern how the value of the corresponding object is formatted.

In example (10) `Console.WriteLine("y={0,9:F3};  d={1,8:F4}", y, d)` outputs two variables - `y, d`. They are numbered by default – the first goes as 0 (for variable `y`) and the second as 1 (for variable `d`). Inside the quotation marks there is the text displayed as it is. The code `("y={0,9:F3}", y)` displays (**y=**) and provides its value. To provide the proper output string format for the value of *variable* `y,` the string format is in curly brackets

`({0,9:F3})`. The `0` is the number of variable, and then after coma its format goes. Here `F3` indicates, that *fixed-point format* is used (F) with 3 digits after the decimal separator; 9 provides the total length of displayed symbols. The resulted y value is 2265.05130421377. On the screen it will appear in the following format (Fig. 1.9):

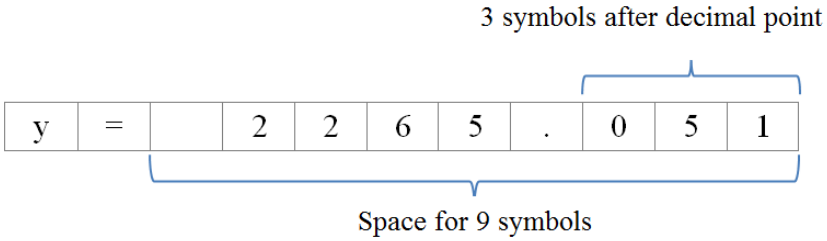


Figure 1.9 - Demonstration of `Console.WriteLine("y={0,9:F3}; y)` format

Totally 9 symbols for the number with decimal separator (if it is less then it provides empty space in the beginning) and 3 digits after the decimal separator.

11) `Console.WriteLine("Press Enter");`

The output of the program is produced by the **WriteLine** method of the **Console** class in the **System** namespace. Here on the screen it displays the text, entered inside quotation marks (Press Enter).

12) `Console.ReadLine();`

Console.ReadLine Method reads the next line of characters from the standard input stream. **Console.Read()** reads only the next character from standard input, and **Console.ReadLine()** reads the next line of characters from the standard input stream. Read() and ReadLine() is used the enter key for exit. If we use ReadLine() or Read() we need press *Enter* button to come back to code.

Solution 2.

It creates the console application with data input through the variables in programming code and the results output in command line.

1. Repeat the steps 1–4 from *Solution 1* to create the console application. The file name will be *Solution 2*.
2. Enter the following C# source code for solving the Task 1.1:

```
using System;
namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            // Solution 2
            double a, b, x, y, d;
            a = 3.2; b = 17.5; x = -4.8;
            y = b * Math.Pow(Math.Tan(x), 2) - a /
                Math.Pow(Math.Sin(x / a), 2);
            d = a * Math.Exp(-Math.Sqrt(a)) * Math.Cos(b*x/a);
            Console.WriteLine("y={0,9:F3}; d={1,8:F4}", y, d);
            Console.WriteLine("Press Enter");
            Console.ReadLine();
        }
    }
}
```

3. Run the program.
Press Start button (Fig. 1.6) or F5.
It will give the results of calculations displayed in command line (Fig. 1.10).

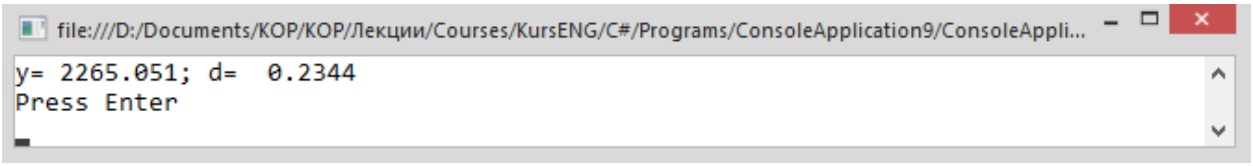


Figure 1.10 - Command line with the results of calculations

4. Save your Project (Fig. 1.8) as *Solution2*.
System will save your project: *D:\ Documents \O65\Programs\Solution1\ Solution2.sln*.

The explanation of the C# code, which is used in the example (Solution 2):

1) `double a, b, x, y, d;`
`a = 3.2; b = 17.5; x = -4.8;`

Declaration statements are used to declare local variables and constants. Firstly it is needed to declare the variables, which will be used in the program, specifying their type. Here the variables `a, b, x, y, d` of `double` type are

declared. Then in the program their values are assigned `a = 3.2; b = 17.5; x = -4.8`.

In Microsoft Visual Studio 2012 the **decimal separator is point < . >**, but the input and output in command line uses the decimal separator from the system. If in operating system the installed decimal separator is coma < , >, then the input through the console should be with coma and the provided output will be with coma.

Solution 3.

It creates the console application with data input from keyboard in the command line and the results output in command line.

1. Repeat the steps 1-4 from *Solution 1* to create the console application. The file name will be *Solution 2*.

The flowchart of the Solution 3 is presented in Fig. 1.11.

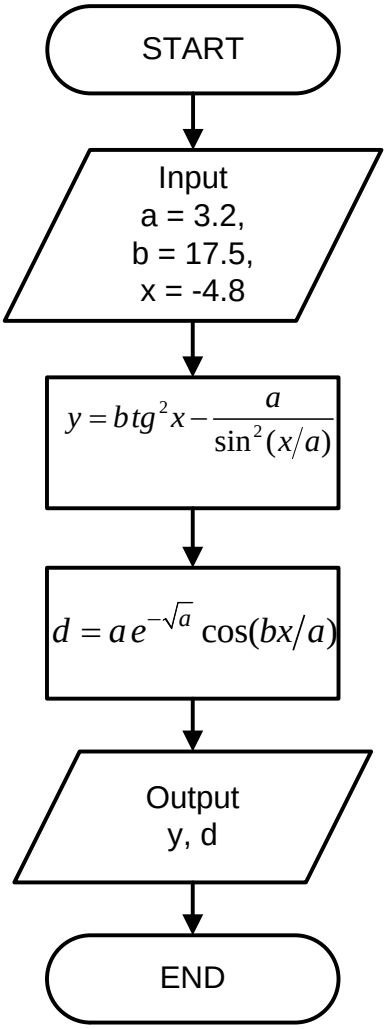


Figure 1.11 - Flowchart of Solution 3

2. Enter the following C# source code for solving the Task 1.1:

```
using System;
namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            //Prgram 3
            double a, b, x, y, d;
            Console.WriteLine(" Input a, b, x: ");
            a = Convert.ToDouble(Console.ReadLine());
            b = Convert.ToDouble(Console.ReadLine());
            x = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine();
            y = b * Math.Pow(Math.Tan(x), 2) - a /
                Math.Pow(Math.Sin(x / a), 2);
            d = a * Math.Exp(-Math.Sqrt(a)) * Math.Cos(b *x/a);
            Console.WriteLine("y={0,9:F3}; d={1,8:F4}", y, d);
            Console.WriteLine("Press Enter");
            Console.ReadLine();
        }
    }
}
```

3. Run the program.

Press Start button (Fig. 1.6) or F5.

When entering the `a, b, x` values press ENTER after each number. The results of calculations are listed in Fig. 1.12.

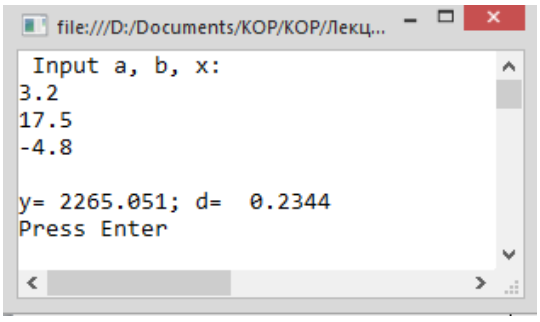


Figure 1.12 - The calculation results (Solution 3)

The explanation of the C# code, which is used in the example (Solution 3):

```
1) Console.WriteLine(" Input a, b, x: ");
```

Because of the `using` directive, the program can use **Console.WriteLine** as shorthand for **System.Console.WriteLine**. **Console.WriteLine** method writes the specified data, followed by the current line terminator, to the standard output stream. In this example the output of the program is produced by the WriteLine method of the Console class in the System namespace. This class is provided by the .NET Framework class libraries, which, by default, are automatically referenced by the Microsoft C# compiler. It will show the following information on the screen: " Input a, b, x:". Then the carriage moves to the next line.

```
2) a = Convert.ToDouble(Console.ReadLine());
   b = Convert.ToDouble(Console.ReadLine());
   x = Convert.ToDouble(Console.ReadLine());
```

The assignment of values to variables `a, b, x`. The values are entered from keyboard.

After running the code, user will see the message on the screen: " Input a, b, x:". Then he inputs the values from the keyboard. The input of each single value should be finished by pressing Enter.

`Console.ReadLine()`

Console.ReadLine method reads the next line of characters from the standard input stream. Here firstly it reads `a` value from the keyboard, then `b` and `x`.

`Convert.ToDouble()`

Convert Class converts a base data type to another base data type.

Convert.ToDouble Method (String) converts the specified string representation of a number to an equivalent double-precision floating-point number.

In this example the values are entered from keyboard, and initially are String type. As we declared variables as `double` type, it is needed to assign the values for the variables in proper type. The assigned values should be `double`. So, it is needed to convert the string entered from keyboard to `double` type. The **Convert.ToDouble Method (String)** is used.

Solution 4.

It creates the Windows Application. Data input is done using the text boxes and the results output using the label form.

The flowchart of Solution 4 is the same as for solution 3 and presented in Fig. 1.11.

To create a C# windows application

1. Open Microsoft Visual Studio 2012.
Пуск /Программы /Программирование /Microsoft Visual Studio 2012 / ∞ Microsoft Visual Studio 2012
The start window of the program appears (Fig. 1.2).

2. On the **File** menu, click **New Project**.
The **New Project** dialog box appears. This dialog box lists the different default application types that Visual C# Express Edition can create (Fig. 1.13).

3. Select **Windows Forms Application** as your project type and change the name of your application to ***Solution4***.
The application environment should be **.NET Framework 4**.
Language (left column): **C#**.
Select **Windows Application** visual C# to create new Windows application.

The default name and location should be fine, but you can always enter a new path if you want. The default name is *WindowsApplication1*, you need to change it for ***Solution4***, and the path: ***D:/Documets/O65/Programs***.

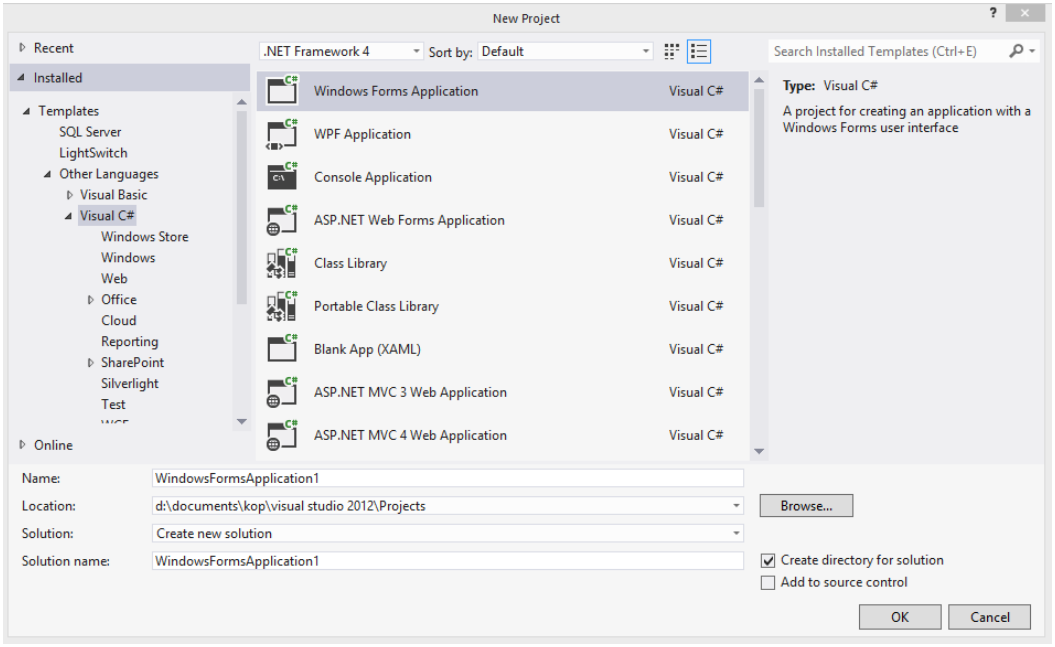


Figure 1.13 - **New project** dialog box

4. Click **OK**.

Visual Studio 2012creates a new folder for your project named after the project title. It also opens the main Visual C# Express Edition window with the *Design* layout. The initial form (default name is *Form1*) for Windows Forms Application opens (see Fig. 1.14).

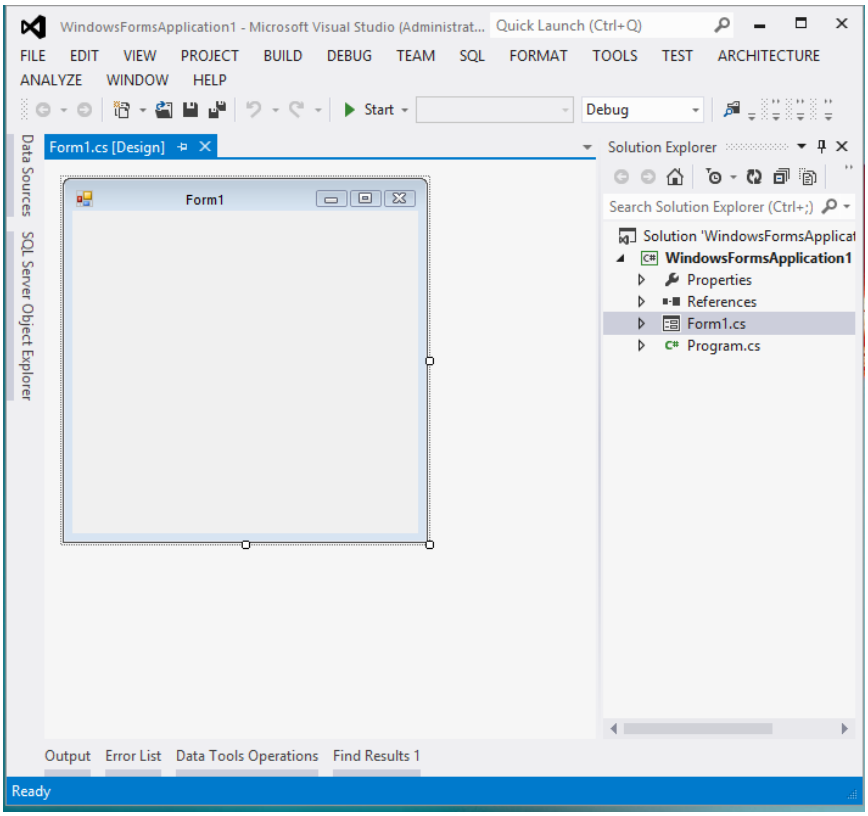


Figure 1.14 - New form for Windows Forms Application (*design* mode)

5. Add the **Toolbox**, **Properties** and **Solution Explorer** windows to work with Form1.

It can be done in main Menu → View → **Toolbox**; main Menu → View → **Solution explorer**; main Menu → View → **Properties** (Fig. 1.15). All these windows should be on the screen (Fig. 1.16).

Solution explorer Window

Solution Explorer (Fig. 1.17) provides you with an organized view of your projects and their files as well as ready access to the commands that pertain to them. A toolbar associated with this window offers commonly used commands for the item you highlight in the list. To access **Solution Explorer**, select **Solution Explorer** on the **View** menu.

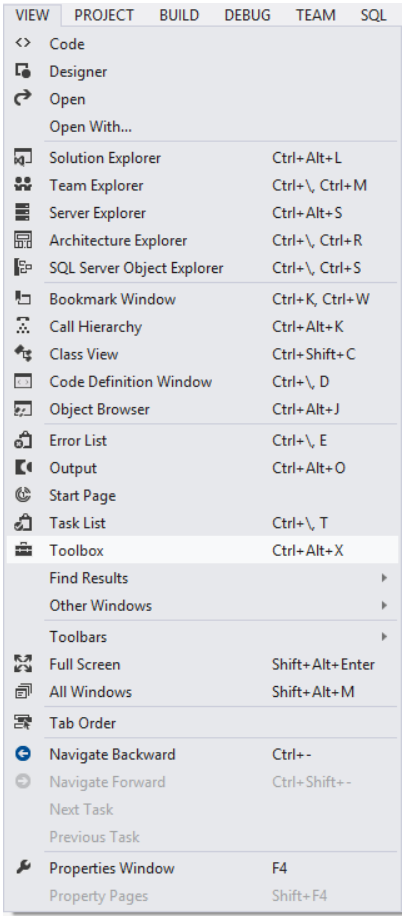


Figure 1.15 - **View** menu

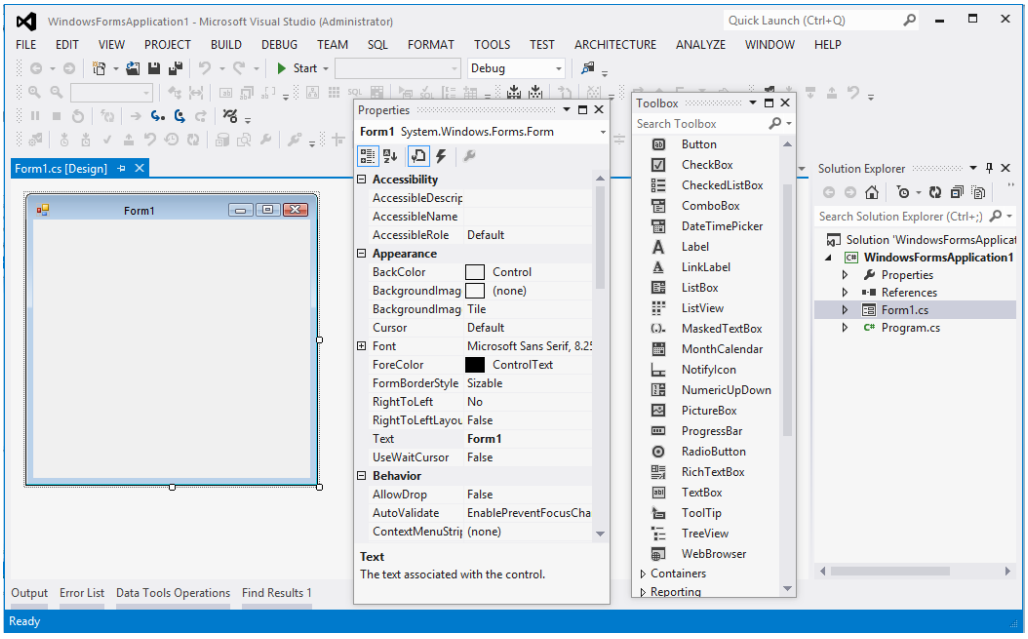


Figure 1.16 - Visual C# environment for windows applications

The management of items displayed in **Solution Explorer** is based on an item's relationship with project and solution containers. Items can be related in the following ways:

- As *project items*, which appear under a project folder in **Solution Explorer**, for example, forms, source files, and classes.
- As *solution items*, which appear in the Solution Items folder of **Solution Explorer**.
- As *miscellaneous files*, which are files that are not associated with either a project or a solution and are displayed in the Miscellaneous Files folder.

Solution Explorer is flexible in that it allows you to work independently of a project; you can edit and create files without a project. **Solution Explorer** displays these files in the Miscellaneous Files folder. You can also work on files that are associated only with the solution. These items are displayed in the Solution Items folder.

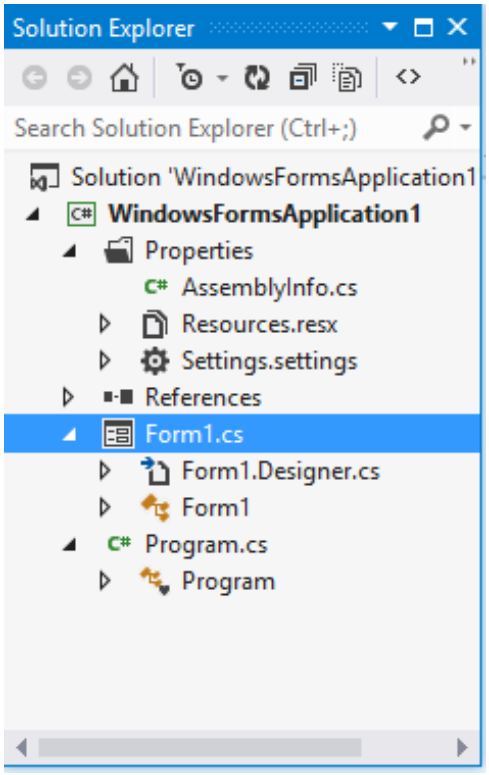
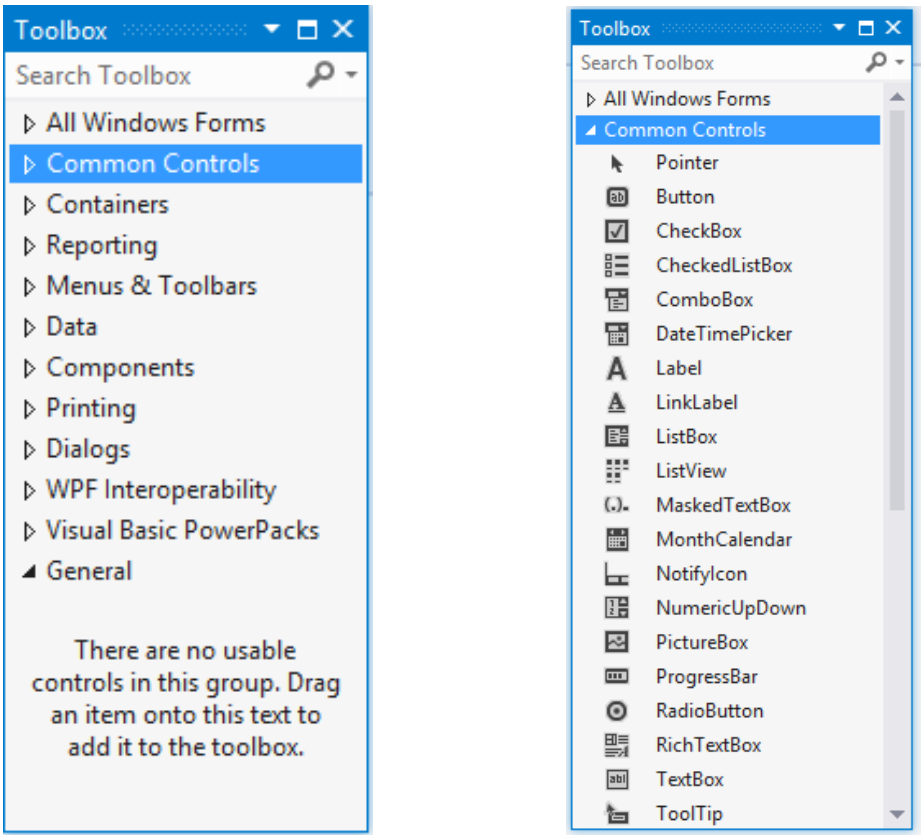


Figure 1.17 - **Solution Explorer**

Toolbox Window

The **Toolbox** (Fig. 1.17) displays icons for controls and other items that you can add to Visual Studio projects. To open the **Toolbox**, click **Toolbox** on the **View** menu.

Every **Toolbox** icon can be dragged to a design view or pasted in a code editor in the Visual Studio integrated development environment (IDE). Either action adds the fundamental code to create an instance of the **Toolbox** item in the active project file.



a) b)

Figure 1.17 - **Toolbox**: a) All **Toolbox** tabs for .NET Framework 4; b) Opened **Common Controls** tab with its components

The **Toolbox** only displays items appropriate to the type of file you are working in. The collection of available controls also depends on the .NET Framework version your project targets. If your project requires a control that is not supported by the Client Profile, you can set your project to target .NET Framework 4 by editing the project properties.

Properties Window

Use this window to view and change the design-time properties and events of selected objects that are located in editors and designers. You can also use the **Properties** window (Fig. 1.18) to edit and view file, project, and solution properties. **Properties** Window is available from the **View** menu.

The **Properties** window displays different types of editing fields, depending on the needs of a particular property. These edit fields include edit boxes, drop-down lists, and links to custom editor dialog boxes. Properties shown in gray are read-only.

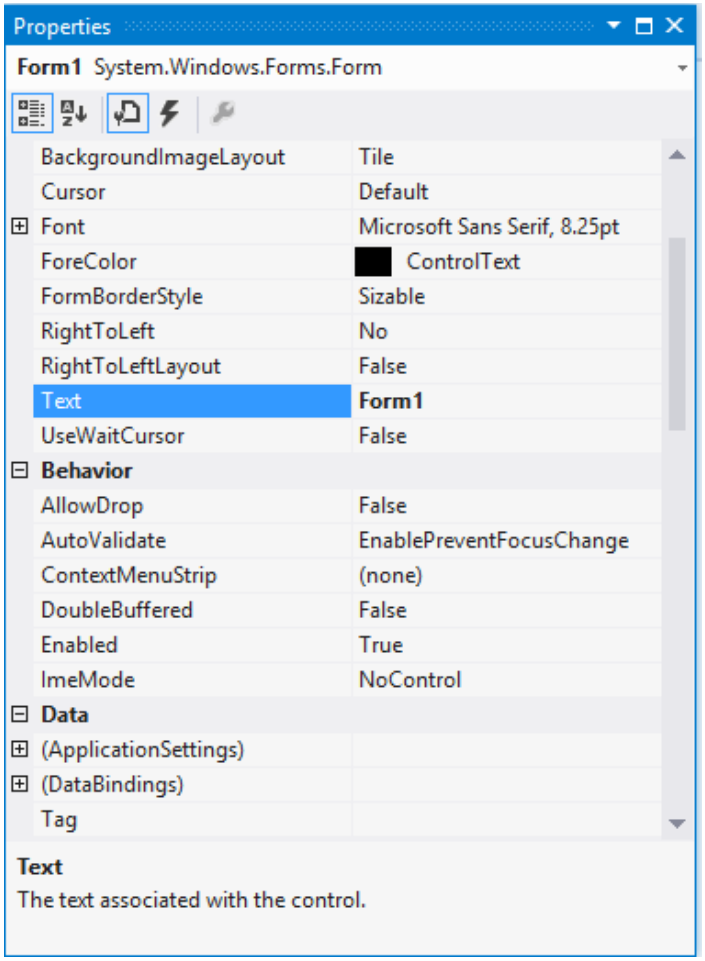


Figure 1.18 - **Properties** Window

6. Open the *Common controls* tab in **Toolbox** window and drag the following objects to the *Form 1*: **Label** (4 pieces); **TextBox** (3 pieces); **Button**. Allocate them in *Form 1* according to Fig. 1.19.

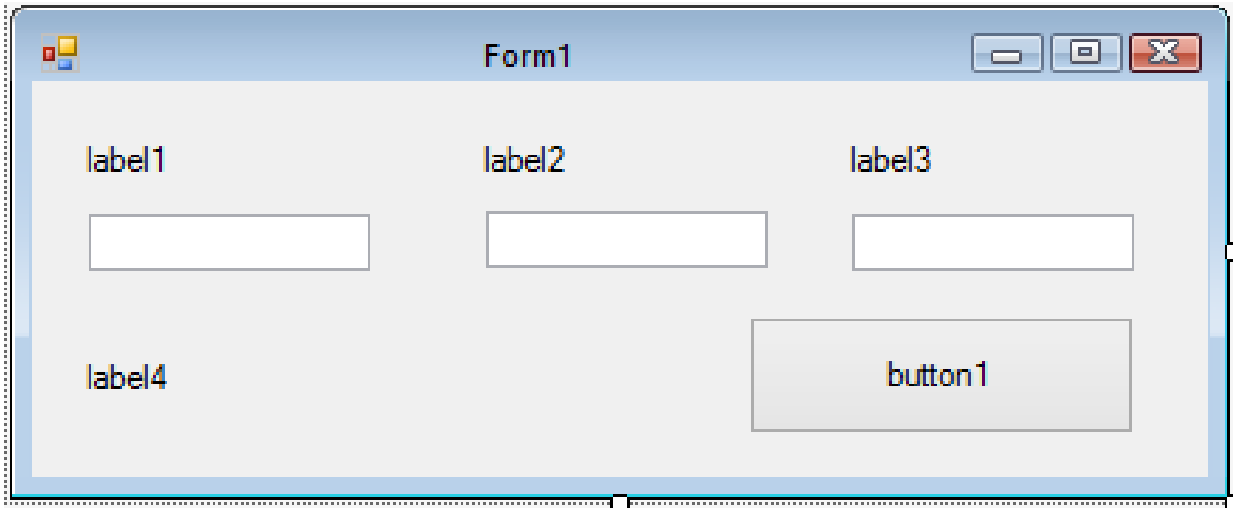


Figure 1.19 - Allocation of the objects used in *Form 1*

7. Change the properties of these objects in **Properties** Window as listed in Table 1.5.

Table 1.5 - The properties of the objects from *Form 1*

Object	The default name of the object (value of Name property field)	Property	New value
1	2	3	4
Form	Form1		
Label	label1	Text	a =
Label	label2	Text	b =
Label	label3	Text	x =
Label	label4		
Select objects label1, label2, label3, label4 (by pressing [Shift] +selecting by mouse) and change the properties of the fields TextAlign and Font		TextAlign	TopCenter
		Font	Times New Roman, Regular, 12
TextBox	textBox1	Name	txta
		Text	3.2
TextBox	textBox2	Name	txtb
		Text	17.5
TextBox	textBox3	Name	txtx
		Text	-4.8
Select objects TextBox1, TextBox2, TextBox3 and change the properties of the fields TextAlign and Font		TextAlign	Center
		Font	Times New Roman, Regular, 12
Button	button1	Name	cmdStart
		Text	Calculate
		Font	Times New Roman, Bold, 12

The *Form 1* will appear in view presented in Fig. 1.20.

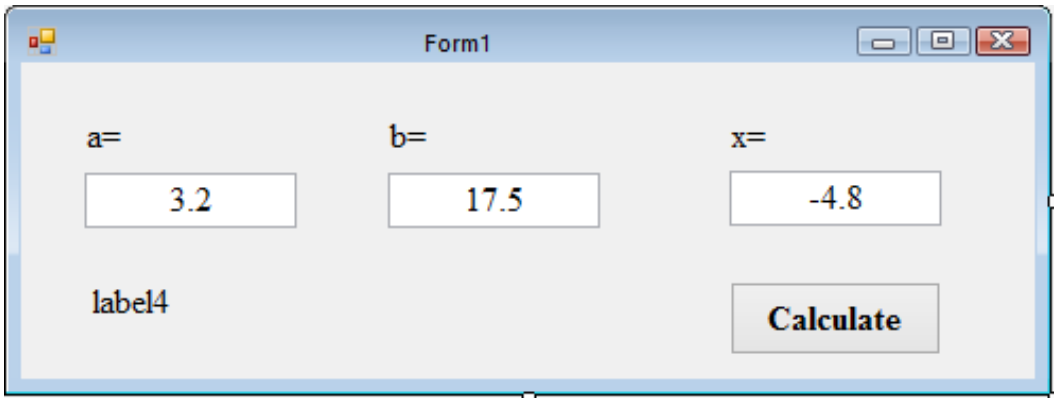


Figure 1.20 - Objects in *Form 1* after changing their properties

When giving the names to objects for the clear understanding of the code it is better to use the common rules. The name of the object can be presented in the following format: PrefixName, where Prefix contains the information about the class of the object, Name is any name given by user and logically connected to the code and names of variables. The following prefixes are usually used for the objects' names:

Prefix	**Control element**	**Example**
txt	TextBox	txtA
cmd	Button	cmdStart
pic	PictureBox (figure)	picFig

8. Write the C# code for solving the Task 1.1, working when the created **Calculate** button is pressed.

Events enable a *class* or *object* to notify other classes or objects when something of interest occurs. The class that sends (or *raises*) the event is called the *publisher* and the classes that receive (or *handle*) the event are called *subscribers*.

In a typical C# **Windows Forms application**, you subscribe to events raised by controls such as buttons and list boxes. You can use the Visual C# integrated development environment (IDE) to browse the events that a control publishes and select the ones that you want to handle. The IDE automatically adds an empty event handler method and the code to subscribe to the event.

All events in the .NET Framework class library are based on the EventHandler delegate, which is defined as follows:

```
public delegate void EventHandler(object sender, EventArgs e);
```

In the example we created a button “Calculate”, which should do the calculations when user presses it. Here the event is the action, when user presses the “Calculate” button. To add the programming code, it is needed to press “Calculate” button two times on Form1. It will lead to the code window in Visual C# IDE (Fig. 1.21).

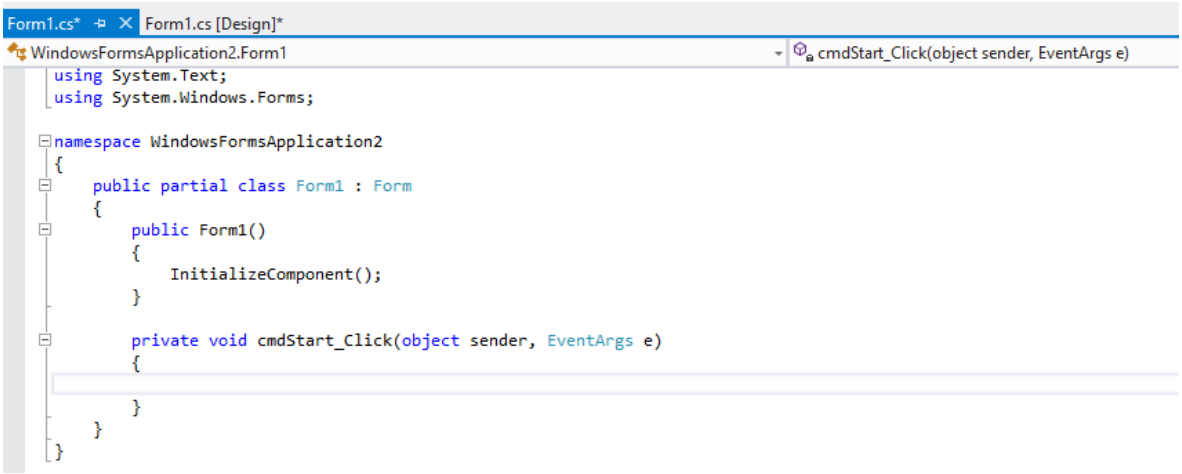


Figure 1.21 - Code environment for *Form 1* in Windows application

The code for event is defined as follows:

```
private void cmdStart_Click(object sender, EventArgs e)
```

Here it is needed to provide the code, which will be done with this event. In our example we need to calculate values `y, d` given in Task 1.1.

Enter the following C# source code:

```
using System;
using System.Windows.Forms;
namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void cmdStart_Click(object sender, EventArgs e)
        {
```

```
            double a, b, x, y, d;
            a = Convert.ToDouble(txta.Text);
            b = Convert.ToDouble(txtb.Text);
            x = Convert.ToDouble(txtx.Text);
            y = b * Math.Pow(Math.Tan(x), 2) - a /
                Math.Pow(Math.Sin(x / a), 2);
            d = a * Math.Exp(-Math.Sqrt(a)) * Math.Cos(b*x/a);
            label4.Text = "y = " + y.ToString("F3") +
                          "      d = " + d.ToString("F4");
        }
    }
}
```

9. Run the program.
Press Start button (Fig. 1.6) or F5.
It will give the results of calculations presented in Fig. 1.22.

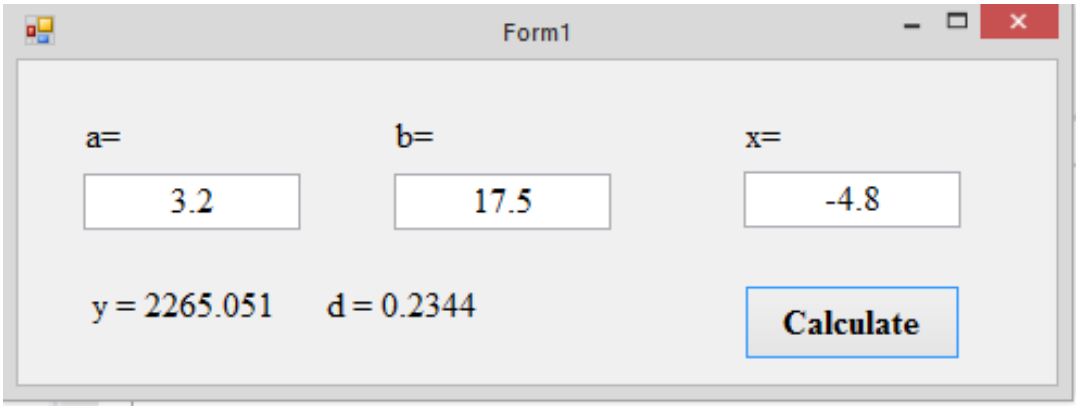


Figure 1.22 - Window with the results of calculations

10. Close the window with the results or go to DEBUG → **Stop Debugging** (to stop the program debugging process).

11. Save your Project by clicking FILE → **Save all**.

System will save your project: *D:\ Documents \O65\Programs\Solution4\ Solution4.sln*.

The explanation of the C# code, which is used in the example (Solution 4):

1) `using System.Windows.Forms;`

The **System.Windows.Forms** namespace contains classes for creating Windows-based applications that take full advantage of the rich user interface features available in the Microsoft Windows operating system.

Table 1.6 shows the classes in **System.Windows.Forms** namespace grouped into categories.

Table 1.6 - The classes in **System.Windows.Forms** namespace.

Class category	**Details**
1	2
Control, User Control, and Form	Most classes within the System.Windows.Forms namespace derive from the *Control* class. The *Control* class provides the base functionality for all controls that are displayed on a *Form*. The *Form* class represents a window within an application. This includes dialog boxes, modeless windows, and Multiple Document Interface (MDI) client and parent windows.
Components	Besides controls, the System.Windows.Forms namespace provides other classes that do not derive from the *Control* class but still provide visual features to a Windows-based application.
Common Dialog Boxes	Windows provides several common dialog boxes that you can use to give your application a consistent user interface when performing tasks such as opening and saving files, manipulating the font or text color, or printing. The *OpenFileDialog* and *SaveFileDialog* classes provide the functionality to display a dialog box that lets the user locate and enter the name of a file to open or save.
Menus and Toolbars	Windows Forms contains a rich set of classes for creating your own custom toolbars and menus with modern appearance and behavior.
Controls	The System.Windows.Forms namespace provides a variety of control classes that you can use to create rich user interfaces. Some controls are designed for data entry within the application, such as *TextBox* and *ComboBox* controls. Other controls display application data, such as *Label* and *ListView*. The namespace also provides controls for invoking commands within the application, such as *Button*. Additionally, you can use the *PropertyGrid* control to create your own Windows Forms Designer that displays the designer-visible properties of the controls.

Table 1.6 (continuation)

1	2
Layout	Several important classes in Windows Forms help control the layout of controls on a display surface, such as a form or control. *FlowLayoutPanel* lays out all the controls it contains in a serial manner, and *TableLayoutPanel* lets you define cells and rows for laying out controls in a fixed grid. *SplitContainer* divides your display surface into two or more adjustable parts.
Data and Data Binding	Windows Forms defines a rich architecture for binding to data sources such as databases and XML files.

```
2) a = Convert.ToDouble(txta.Text);
   b = Convert.ToDouble(txtb.Text);
   x = Convert.ToDouble(txtx.Text);
```

The assignment of values to variables `a, b, x`.

The value from the **Text** property of the object with **txta** name (`txta.Text`) is converted using **Convert.ToDouble Method (String)** to the double-precision floating-point number and assigned to variable `a` (`double`). The **Convert.ToDouble Method (String)** is used, because by default in Visual C# the content in the text field is of **text** type.

```
3) label4.Text = "y = " + y.ToString("F3") + " d = " +
d.ToString("F4");
```

The data output to the **Text** property of object **label4**.

Double.ToString() Method converts the numeric value of this instance to its equivalent string representation. The text inside quotation marks “ ” and without the curly brackets is displayed as it is.

To display the output data in proper format, you need to specify this format inside quotation marks according to the Table 4. Here the selected format is `"F3",` which means that `F (Fixed-point)` format is used with three digits after the decimal separator. Binary `+` operators are predefined for *numeric* and *string* types. For numeric types, `+` computes the sum of its two operands. When one or both operands are of type string, `+` concatenates the string representations of the operands.

Task 1.2. Tasks for laboratory works.

Table 1.7 - Variants for calculation

№	Relations for calculation	Initial data
1	2	3
1	$a = \dfrac{2\cos(x - \pi/6)}{1/2 + \sin^2 y}$ $b = 1 + \dfrac{z^2}{3 + z^2/5}$	$x = 1.426$ $y = -1.22$ $z = 3.5$
2	$\gamma = \left\lvert x^{y/x} - \sqrt[3]{y/x} \right\rvert$ $\psi = (y - x)\dfrac{y - z/(y - x)}{1 + (y - x)^2}$	$x = 1.825$ $y = 18.225$ $z = -3.298$
3	$y = e^{-bt}\sin(at + b) - \sqrt{\lvert bt + a \rvert}$ $s = b\sin(at^2 \cos 2t) - 1$	$a = -0.5$ $b = 1.7$ $t = 0.44$
4	$\omega = \sqrt{x^2 + b} - b^2 \sin^3(x + a)/x$ $y = \cos^2 x^3 - x/\sqrt{a^2 + b^2}$	$a = 1.5$ $b = 15.5$ $x = -2.9$
5	$s = x^3 \mathrm{tg}^2(x + b)^2 + a/\sqrt{x + b}$ $Q = \dfrac{bx^2 - a}{e^{ax} - 1}$	$a = 16.5$ $b = 3.4$ $x = 0.61$
6	$R = x^2(x + 1)/b - \sin^2(x + a)$ $S = \sqrt{xb/a} + \cos^2(x + b)^3$	$a = 0.7$ $b = 0.05$ $x = 0.5$
7	$y = \sin^3(x^2 + a)^2 - \sqrt{x/b}$ $z = \dfrac{x^2}{a} + \cos(x + b)^3$	$a = 1.1$ $b = 0.004$ $x = 0.2$

Table 1.7 (continuation)

1	2	3
8	$f = \sqrt[3]{m\,\mathrm{tg}\,t + \lvert c\sin t\rvert}$ $z = m\cos(bt\sin t) + c$	$m = 2\ ;\quad c = -1$ $t = 1.2\ ;\quad b = 0.7$
9	$y = \ln^3(1 + x^2)$ $F = \sin x^2 \cos\dfrac{7x - 2}{3,75\pi}$	$x = 1.45$
10	$f = \ln(a + x^2) + \sin^2(x/b)$ $z = e^{-cx}\dfrac{x + \sqrt{x + a}}{x - \sqrt{\lvert x - b\rvert}}$	$a = 10.2\ ;\quad b = 9.2$ $x = 2.2\ ;\quad c = 0.5$
11	$a = 1,2c\ ;\quad b = 3c/5$ $k = (a^{3/2} + b^{3/2})/(a^2 - ab)^{3/2}$ $F = \cos(x^2 + 1,43\pi) + x/2$	$c = 2.15$ $x = 2.5$
12	$y = \dfrac{a^{2x} + b^{-x}\cos(a + b)x}{x + 1}$ $R = \sqrt{x^2 + b} - b^2\sin^3(x + a)/x$	$a = 0.3$ $b = 0.9$ $x = 0.61$

Questions to Chapter 1.

1. What is the difference between console and Windows applications?
2. Types in C# language.
3. Explain the method Console.Write(). When is it used?
4. Explain the method Console.WriteLine(). When is it used?
5. Explain the method Console.Read(). When is it used?
6. Explain the method Console.ReadLine(). When is it used?

Chapter 2. Programs with branch structure

Objectives:

1. For console applications
   To study the selection statements `if` and `switch`.
2. For Windows applications
   To study the selection statements `if` and `switch` and the Common Controls elements: Label, Button, TextBox.

Task 2.1. Calculate the value of the expression.

$$y = \begin{cases} x^2 + 5/a, & x < 0 \\ 12 + a, & x \geq 0 \end{cases} \qquad \text{where} \quad a = 5.2; \qquad x = -3; \quad 0; \quad 34.85$$

The flowchart of the solution for Task 2.1 is presented in Fig. 2.1.

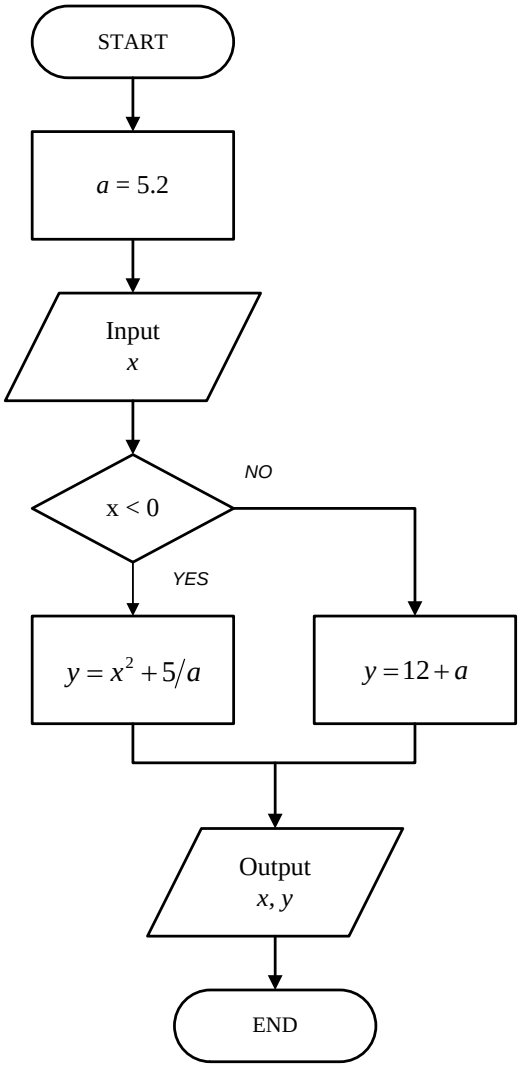


Figure 2.1 - Flowchart of solution for Task 2.1

Solution 1.

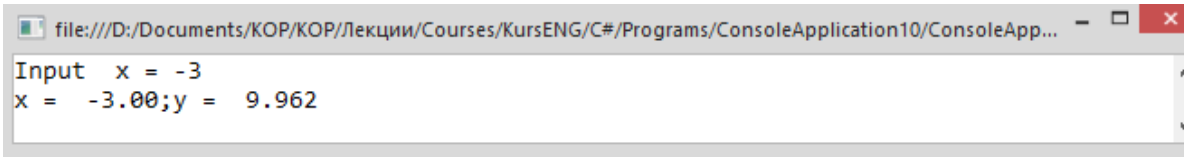
It creates the console application with data input by the user from command line. Do the steps 1 - 4 from Task 1.1, solution 1 to create *Console application*. The file name – *Task2_Solution1* in ***D:/Documents/O65/Programs*** folder.

Enter the following C# source code for solving the Task 2.1:

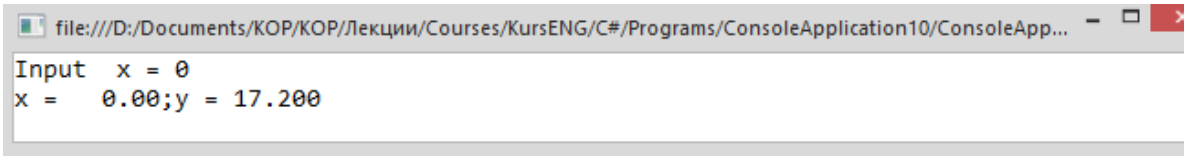
```
using System;
namespace Task2_Solution1
{
    class Program
    {
        static void Main(string[] args)
        {
            double a, x, y;
            a = 5.2;
            Console.Write("Input  x = ");
            x = Convert.ToDouble(Console.ReadLine());
            if (x < 0)
                y = x * x + 5 / a;
            else
                y = 12 + a;
            Console.WriteLine("x = {0,6:F2};y = {1,6:F3}",x,y);
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 2.2.

a)



b)



c)

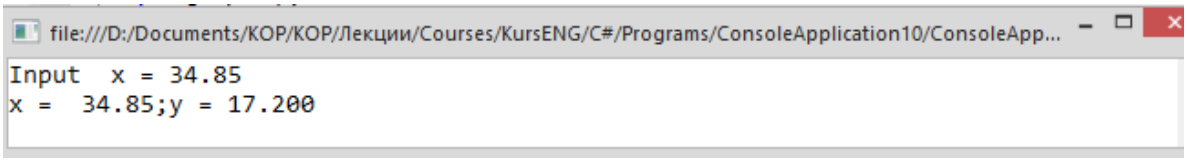


Figure 2.2 - The results of calculations Task 2.1, Solution 1:
a) $x = -3$; b) $x = 0$; c) $x = 34.85$

The explanation of the C# code, which is used in the example (Task 2.1, Solution 1):

`if-else` statement

An `if` statement identifies which statement to run based on the value of a *Boolean* expression. An `if` statement in C# can take two forms:

```
// (1) if-else statement
if (condition)
{
    then-statement;
}
else
{
    else-statement;
}
// Next statement in the program.

// (2) if statement without an else
if (condition)
{
    then-statement;
```

```
}
// Next statement in the program.
```

In an **if-else** statement, if condition evaluates to true, the then-statement runs. If condition is false, the else-statement runs. Because condition can't be simultaneously true and false, the then-statement and the else-statement of an **if-else** statement can never both run. After the then-statement or the else-statement runs, control is transferred to the next statement after the **if** statement.

In an **if** statement that doesn't include an **else** statement, if condition is true, the then-statement runs. If condition is false, control is transferred to the next statement after the **if** statement.

Both the then-statement and the else-statement can consist of a single statement or multiple statements that are enclosed in braces (**{}**). For a single statement, the braces are optional but recommended.

The statement or statements in the then-statement and the else-statement can be of any kind, including another **if** statement nested inside the original **if** statement. In nested **if** statements, each **else** clause belongs to the last **if** that doesn't have a corresponding **else**.

Task 2.2. Calculate the value of the expression.

$$f = \begin{cases} 1 - \cos a - y, & y < 1 \\ (ay + b)/2, & 1 \le y \le 4 \\ a + 1, & y > 4 \end{cases}$$

where $a = 0.2;\ \ b = 0.001;\ \ \ y = 0.5;\ \ 3;\ \ 12$

The flowchart of the solution for Task 2.1 is presented in Fig. 2.3.

Solution 1.

It creates the console application with the nested `if` statements.

Do the steps 1 - 4 from Task 1.1, solution 1 to create *Console application*. The file name – *Task2_2_Solution1* in ***D:/Documents/O65/Programs*** folder.

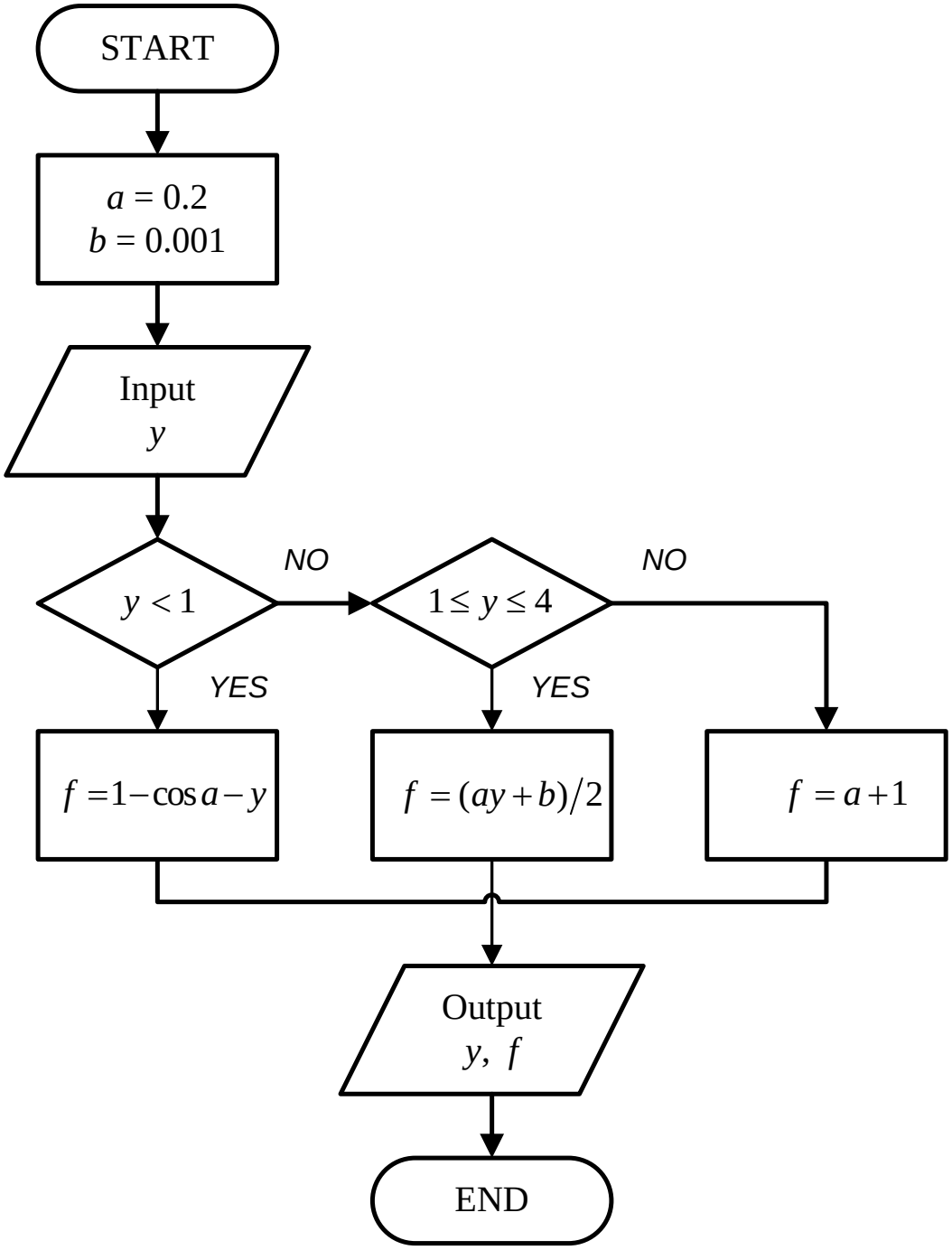


Figure 2.3 - Flowchart of solution for Task 2.2

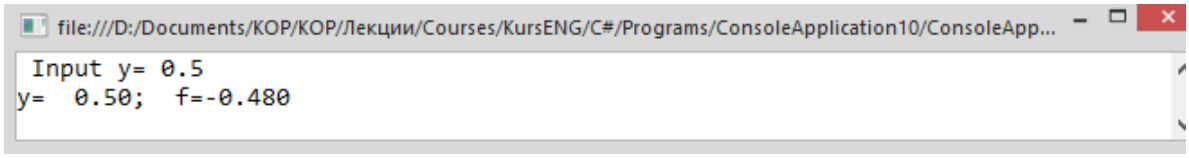
Enter the following C# source code for solving the Task 2.1:

```
using System;
namespace Task2_Solution1
{
    class Program
    {
        static void Main(string[] args)
```

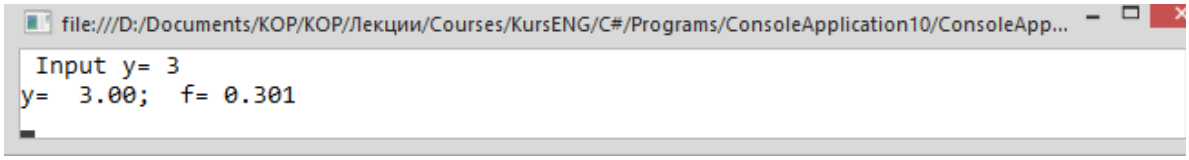
```
        {
            double a, b, y, f;
            a = 0.2; b = 0.001;
            Console.Write(" Input y= ");
            y = Convert.ToDouble(Console.ReadLine());
            if (y < 1)
                f = 1 - Math.Cos(a) - y;
            else if ((1 <= y) && (y <= 4))
                f = (a * y + b) / 2;
            else
                f = a + 1;
            Console.WriteLine("y={0,6:F2};  f={1,6:F3}", y, f);
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 2.4.

a)



b)



c)

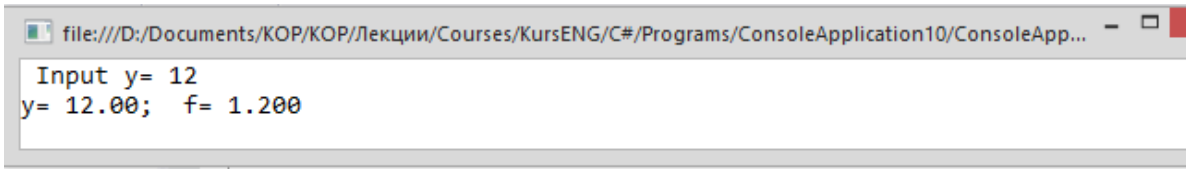


Figure 2.4 - The results of calculations Task 2.1, Solution 1:
a) $y = 0.5$; b) $y = 3$; c) $y = 12$

The explanation of the C# code, which is used in the example (Task 2.2, Solution 1):

1) `(1 <= y) && (y <= 4)`

The mathematical statement $1 \leq y \leq 4$ means that $y \geq 1$ and at the same time $y \leq 4$. In C# programming language two statements are connected using `&&` operator, namely `(1 <= y) && (y <= 4)`. Because in C# notation the comparison goes before logical operations, it is possible to wright without brackets.

Solution 2.

It creates the Windows Application. Data input is done using the text box and the results are displayed in the listBox.

Do the steps 1 - 5 from Task 1.1, Solution 4 to create *Windows application*. The file name – *Task2_2_Solution2* in ***D:/Documents/O65/Programs*** folder.

6. Open the *Common controls* tab in **Toolbox** window and drag the following objects to the *Form 1*: **Label** (2 pieces); **TextBox**; **ListBox**; **Button**. Allocate them in *Form 1* according to Fig. 2.5.

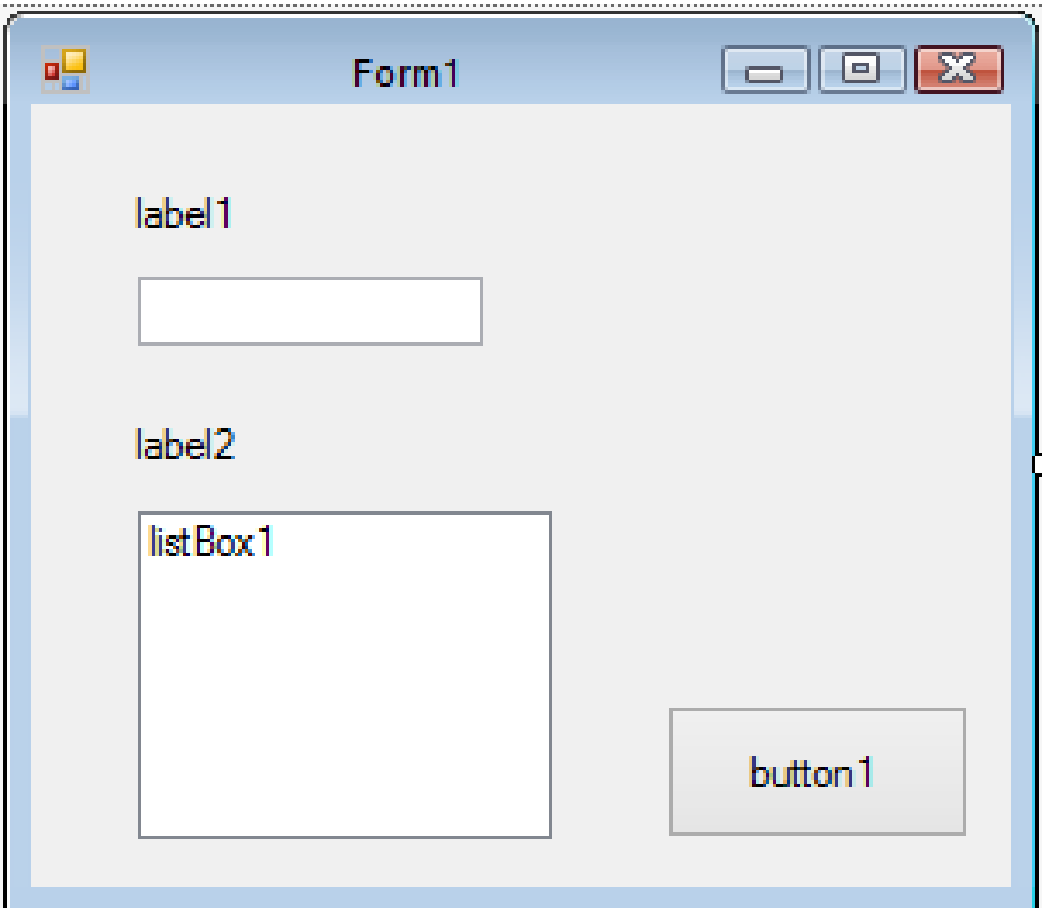


Figure 2.5 - Allocation of the objects used in *Form 1*

7. Change the properties of these objects in **Properties** Window as listed in Table 2.1.

Table 2.1 - The properties of the objects from *Form 1*

Object	The default name of the object (value of Name property field)	Property	New value
Form	Form1		
Label	label1	Text	Input y =
Label	label2	Text	Calculation results
TextBox	textBox1	Name	txty
ListBox	listBox1		
Командная кнопка	button1	Name	cmdStart
		Text	Calculate
		Font	Times New Roman, Bold, 12

The Form 1 will be modified according to Fig. 2.6.

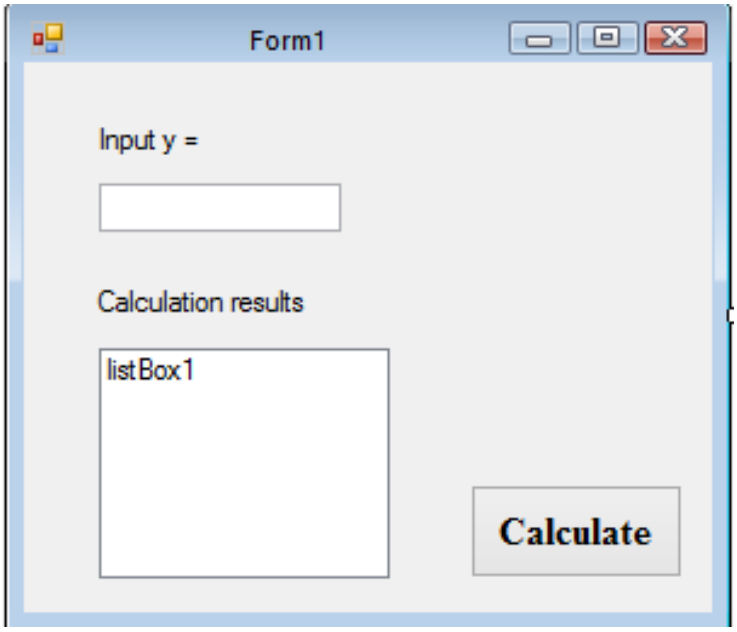


Figure 2.6 - Objects in *Form 1* after changing their properties

8. Write the C# code for solving the Task 2.2, working when the created **Calculate** button is pressed.

```
using System;
using System.Windows.Forms;
namespace WindowsFormsApplication3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void listBox1_SelectedIndexChanged(object
        sender, EventArgs e)
        {

        }
        private void cmdStart_Click(object sender, EventArgs e)
        {
            double a, b, y, f;
            a = 0.2;
            b = 0.001;
            y = Convert.ToDouble(txty.Text);
            if (y < 1)
                f = 1 - Math.Cos(a) - y;
            else if (1 <= y && y <= 4)
                f = (a * y + b) / 2;
            else
                f = a + 1;
            listBox1.Items.Add("y = " + y.ToString("n2") + "\t"
                                + " f = " + f.ToString("n3"));
        }
    }
}
```

9. Run the program.

Press Start button (Fig. 1.6) or F5.

It will give the results of calculations presented in Fig. 2.7.

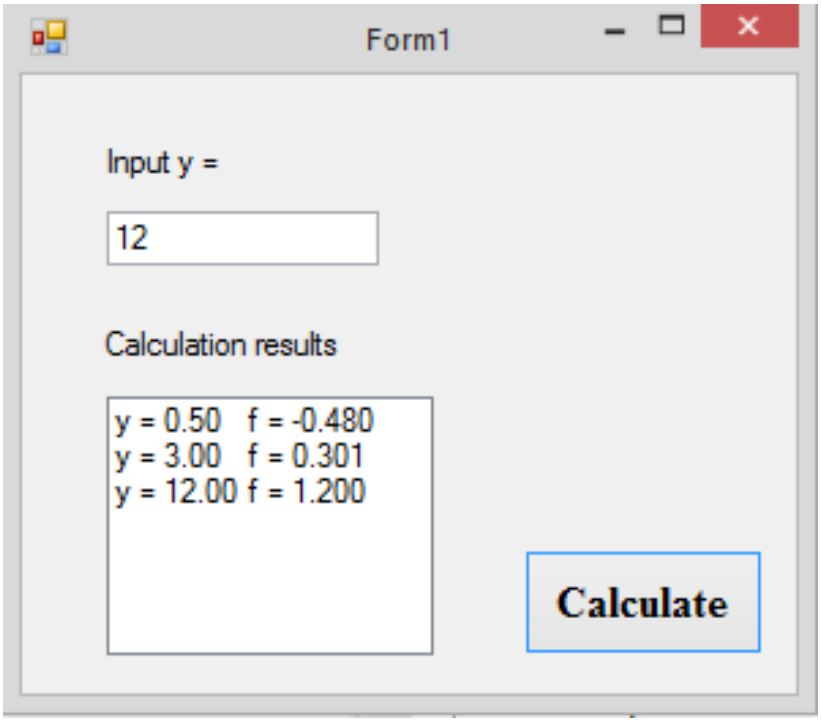


Figure 2.7 - Window with the results of calculations

10. Close the window with the results or go to DEBUG → **Stop Debugging** (to stop the program debugging process).

11. Save your Project by clicking FILE → **Save all**.

System will save your project: *D:\ Documents \O65\Programs\Task2_2_Solution2\ Task2_2_Solution2.sln*.

The explanation of the C# code, which is used in the example (Task 2.2, Solution 2):

ListBox Class represents a Windows control to display a list of items.
ListBox.Items.Add Method adds the elements to the ListBox.

Task 2.3. Calculate the value of the expression.

$$y = \begin{cases} (x \cdot z)^2, & x > 0, z > 0 \\ x^{-z}, & x < 0, z < 0 \\ 5, & \text{in all other cases} \end{cases}$$

where

1) $x = 3; \quad z = 5;$
2) $x = -5; \quad z = -4;$
3) $x = 25; \quad z = -8.$

Solution 1.

It creates the console application with data input by the user from command line.

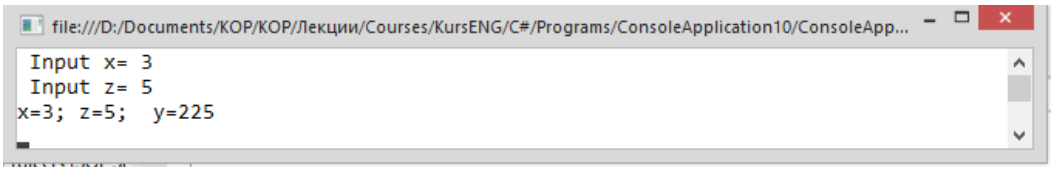
Do the steps 1 - 4 from Task 1.1, solution 1 to create *Console application*. The file name – *Task2_3* in ***D:/Documents/O65/Programs*** folder.

Enter the following C# source code for solving the Task 2.3:

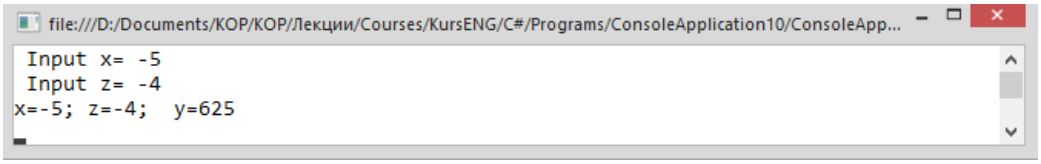
```
using System;
namespace ConsoleApplication4
{
    class Program
    {
        static void Main(string[] args)
        {
            double x, z, y;
            Console.Write(" Input x= ");
            x = Convert.ToDouble(Console.ReadLine());
            Console.Write(" Input z= ");
            z = Convert.ToDouble(Console.ReadLine());
            if (x > 0 && z > 0)
                y = Math.Pow((x * z), 2);
            else if (x < 0 && z < 0)
                y = Math.Pow(x, -z);
            else
                y = 5;
            Console.WriteLine("x={0}; z={1};  y={2} ",x,z,y);
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 2.8.

1)



2)



3)

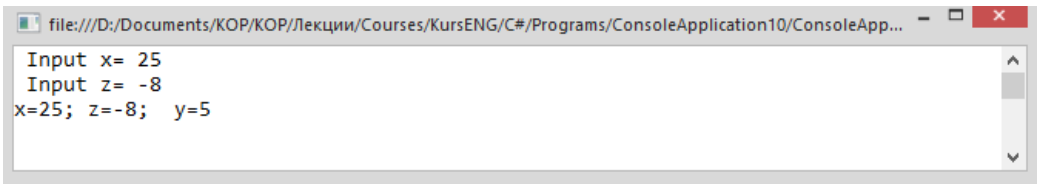


Figure 2.8 - The results of calculations Task 2.3, Solution 1

Task 2.4. Calculate the value of the expression.

$$y=\begin{cases} x, & x=0 \\ x^3+6, & x=1 \\ x^2+2, & 2\le x\le 7 \\ x/3, & x<0 \ \ or \ \ x>100 \\ 5, & in\ \ all\ \ other\ \ cases \end{cases}$$

The flowchart of Solution 3 is presented in Fig. 2.9.

Solution 1.

It creates the console application with data input by the user from command line. Do the steps 1 - 4 from Task 1.1, Solution 1 to create *Console application*. The file name – *Task2_4* in ***D:/Documents/O65/Programs*** folder.

Enter the following C# source code for solving the Task 2.4:

```
using System;
namespace ConsoleApplication5
{
    class Program
```

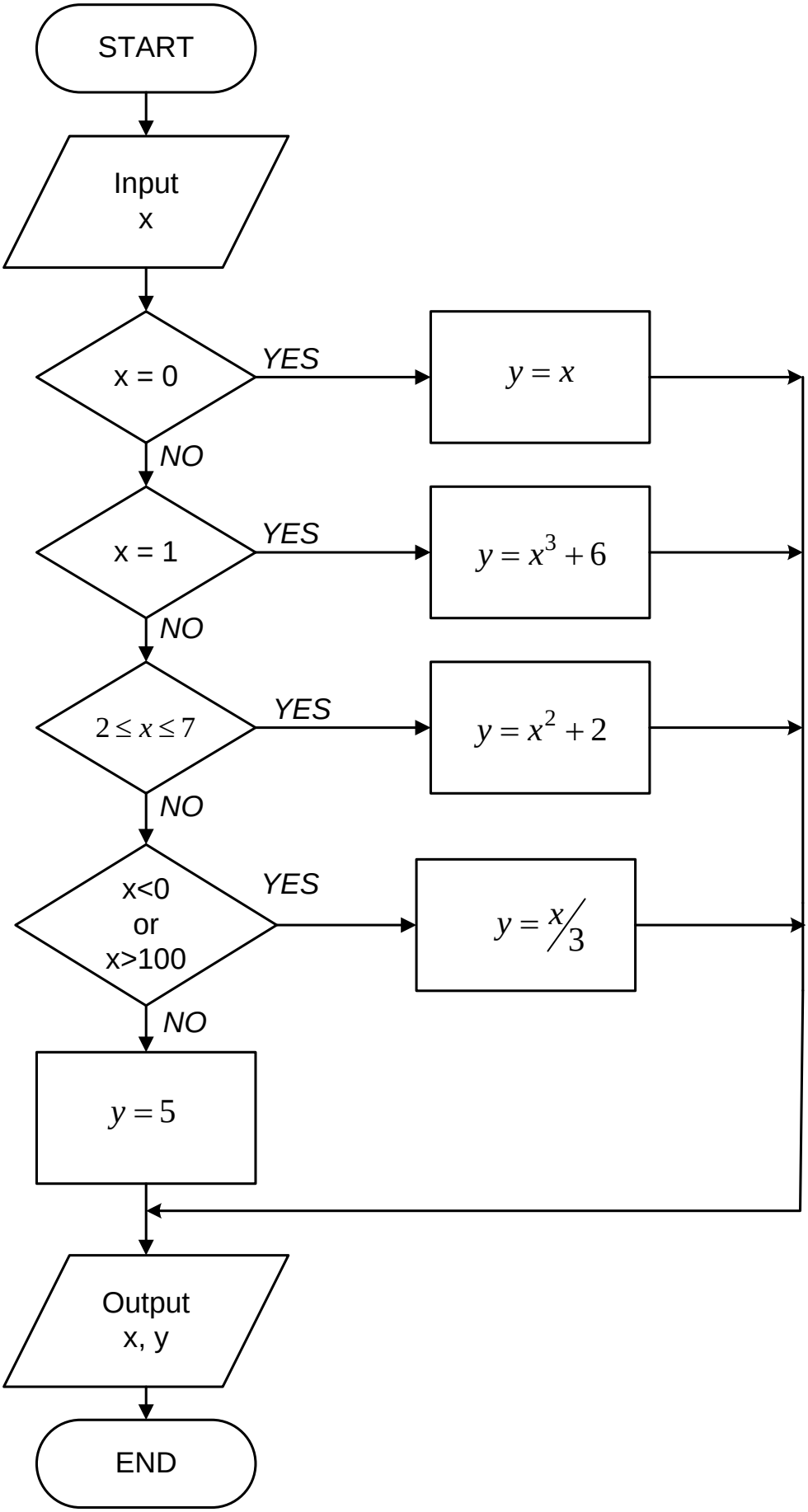
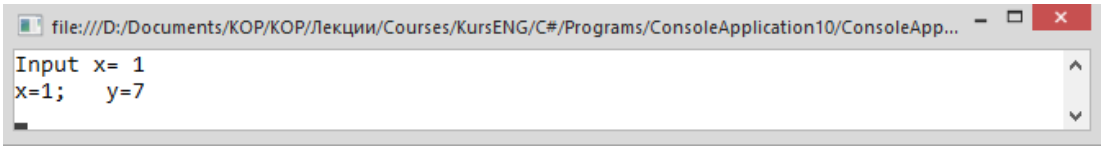


Figure 2.9 - Flowchart of solution for Task 2.4

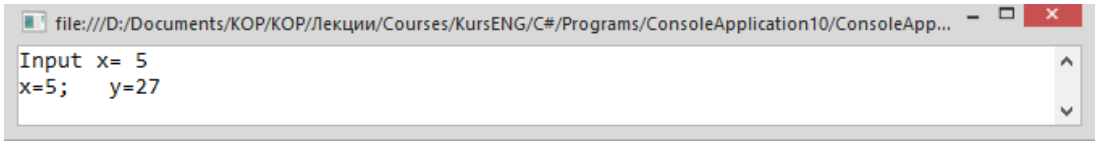
```
    {
        static void Main(string[] args)
        {
            double x, y;
            Console.Write("Input x= ");
            x = Convert.ToDouble(Console.ReadLine());
            if (x == 0)
                y = x;
            else if (x == 1)
                y = x * x * x + 6;
            else if (2 <= x && x <= 7)
                y = x * x + 2;
            else if (x < 0 || x > 100)
                y = x / 3;
            else
                y = 5;
            Console.WriteLine("x={0};   y={1} ", x, y);
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 2.10.

1)



2)



3)

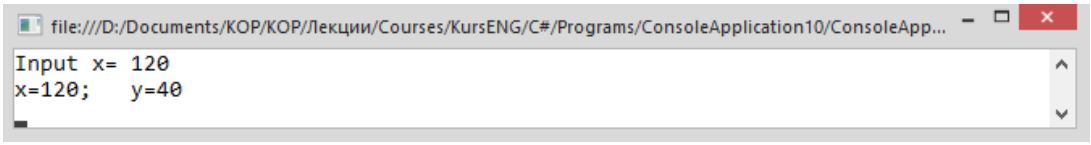


Figure 2.10 - The results of calculations Task 2.4, Solution 1

Task 2.5. Solve the following problem.

Program should provide the names of the week days basing on the day number entered by the user. If user enters inappropriate number, the program should provide the error message. Use `switch` method to solve the task.

The flowchart of the Task 2.5 is presented in Fig. 2.11.

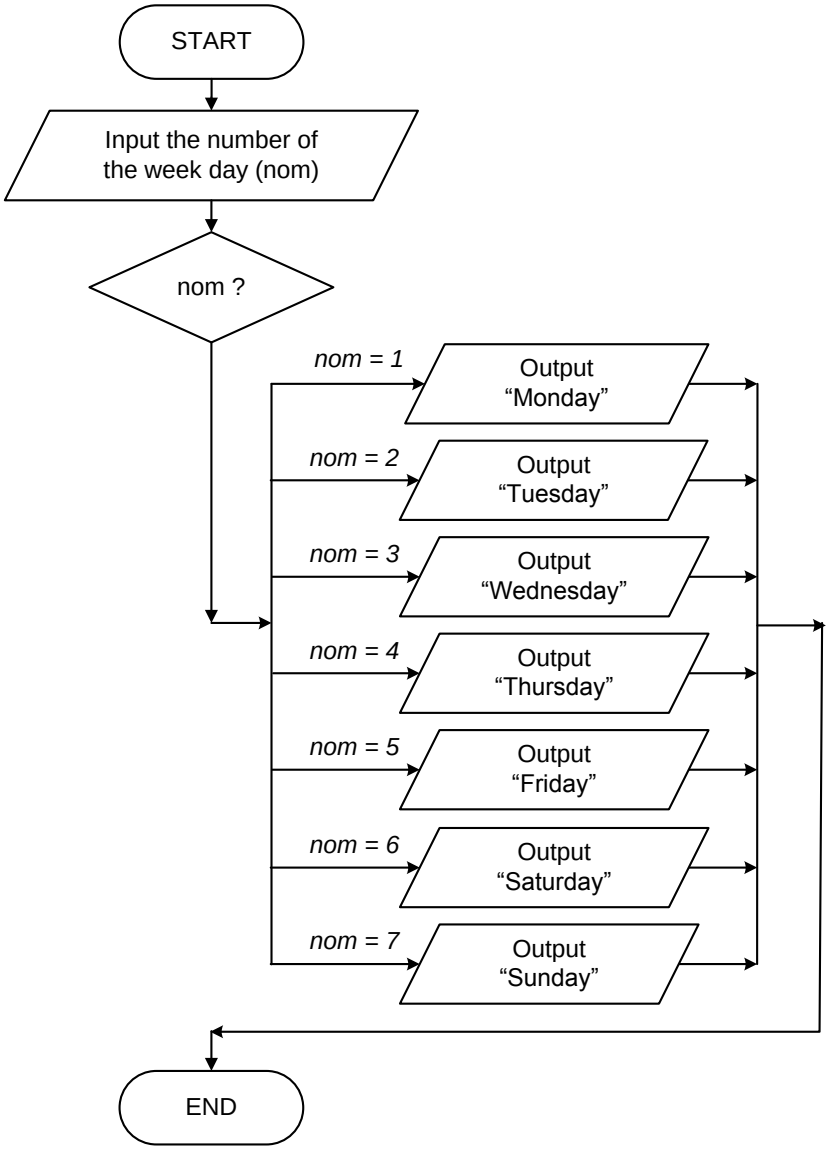


Figure 2.11 - Flowchart of solution for Task 2.5

Solution 1.

It creates the console application with data input by the user from command line.

Do the steps 1 - 4 from Task 1.1, Solution 1 to create *Console application*. The file name – *Task2_5* in ***D:/Documents/O65/Programs*** folder.

Enter the following C# source code for solving the Task 2.5:

```
using System;
namespace ConsoleApplication6
{
    class Program
    {
        static void Main(string[] args)
        {
            int nom;
            Console.Write("Enter the number of the day in the
                           week -> ");
            nom = Convert.ToInt32(Console.ReadLine());
            switch (nom)
            {
                case 1:
                    Console.WriteLine("Monday");
                    break;
                case 2:
                    Console.WriteLine("Tuesday");
                    break;
                case 3:
                    Console.WriteLine("Wednesday");
                    break;
                case 4:
                    Console.WriteLine("Thursday");
                    break;
                case 5:
                    Console.WriteLine("Friday");
                    break;
                case 6:
                    Console.WriteLine("Saturday");
                    break;
                case 7:
                    Console.WriteLine("Sunday");
                    break;
                default:
                    Console.WriteLine("The number should be
                    integer-value from 1 to 7");
```

```
                    break;
                }
                Console.ReadLine();
            }
        }
}
```

The results of calculations are presented in Fig. 2.12.

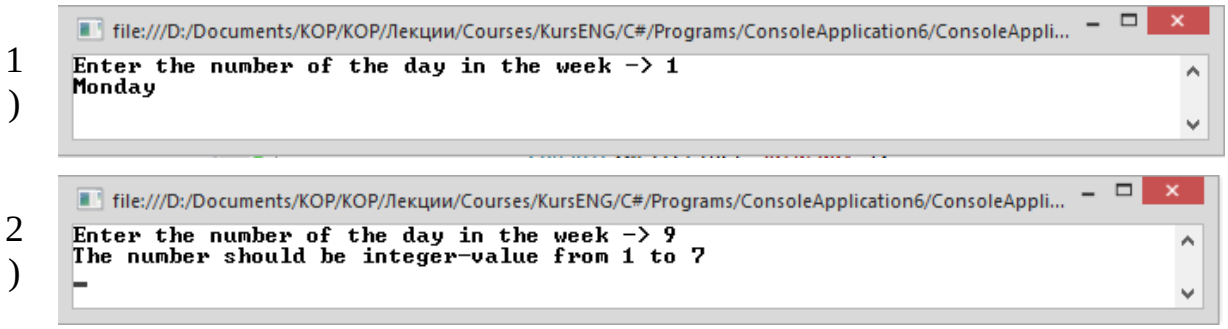


Figure 2.12 - The results of calculations Task 2.5, Solution 1

Task 2.6. Solution of the quadratic equation.

Program should provide the roots of the quadratic equation of the following form: $ax^2 + bx + c = 0$. The user enters the constants a, b, c from the keyboard.

The flowchart of the solution is presented in Fig. 2.13.

Solution 1.

It creates the console application with data input by the user from command line.

Do the steps 1 - 4 from Task 1.1, solution 1 to create *Console application*. The file name – *Task2_6* in ***D:/Documents/O65/Programs*** folder.

Enter the following C# source code for solving Task 2.6:

```
using System;
namespace Task2_6
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
            double a, b, c, D, x1, x2;
            Console.WriteLine(" Input a, b, c:  ");
            a = Convert.ToDouble(Console.ReadLine());
            b = Convert.ToDouble(Console.ReadLine());
            c = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine();
            D = b * b - 4 * a * c;
            if (D >= 0 && a != 0)
            {
                x1 = (-b + Math.Sqrt(D)) / (2 * a);
                x2 = (-b - Math.Sqrt(D)) / (2 * a);
                Console.WriteLine("x1={0};   x2={1}", x1, x2);
            }
            else
            {
                Console.WriteLine(" No solution ");
            }
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Figure 2.14.

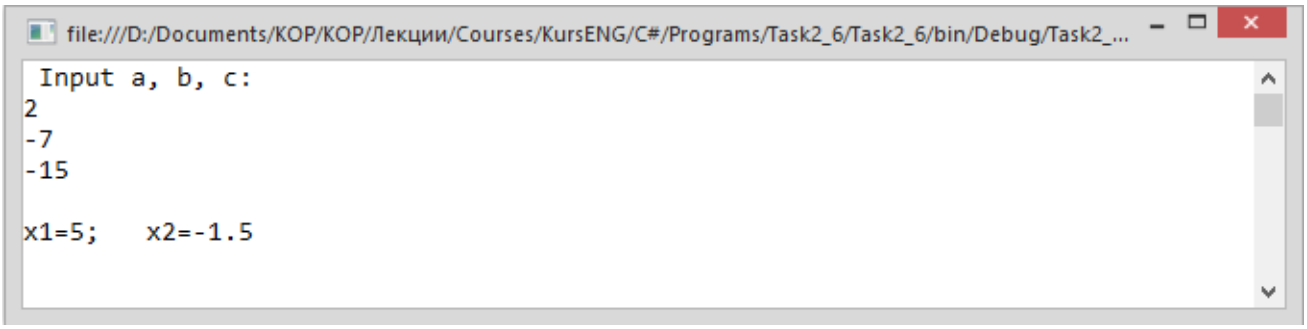


Figure 2.14 – The results of calculations Task 2.6, Solution 1

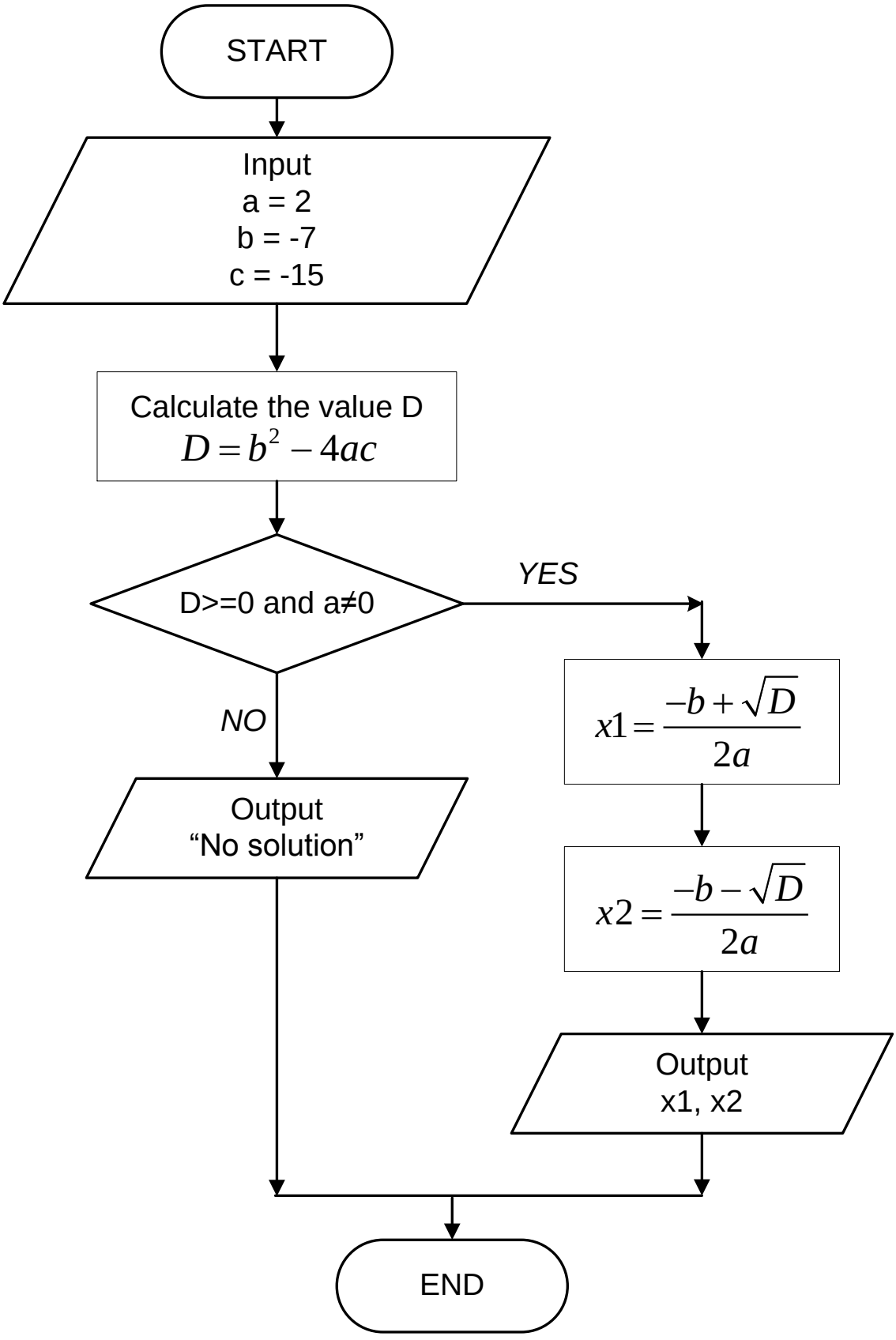


Figure 2.13 - Flowchart of solution for Task 2.6

Task 2.7. Tasks for laboratory works.

Table 2.2 - Variants for calculation

№	Relations for calculation	Initial data
1	2	3
1	$z=\begin{cases}\sqrt{\lvert x^2+2x+y\rvert}, & x<0,\ y<0\\ x+y, & x>0,\ y>0\\ \lvert x\rvert-\lvert y\rvert, & \textit{in other cases}\end{cases}$	$x=-0.5;\ \ y=-2.5$ $x=2.31;\ \ y=4.2$ $x=5;\ \ y=-7$
2	$y=\begin{cases}\sin^2 x, & x>0\\ \ln^3(1+x^2), & x<0\\ x+c, & x=0\end{cases}$	$c=1.57$ $x=3;\ -2.5;\ 0$
3	$g=\begin{cases}y^3-0.3, & y<0\\ 0, & 0\le y\le 1\\ y^2+y, & y>1\end{cases}$	$y=z+2;$ $z=-3;-1.5;\ 0$
4	$c=\begin{cases}(xy)^z, & y>0,\ z>0\\ (x+5y+z)^2, & y<0,\ z<0\\ 0, & \textit{in other cases}\end{cases}$	$y=1;-0.5;\ 0$ $z=0.5;-1.5;\ 3$ $x=0.5$
5	$m=\begin{cases}(t^2+1)bc, & t<0\\ 1/(t+1), & 0\le t\le 4\\ \sin(t+1)^2, & t\ge 4\end{cases}$	$b=0.5;\ \ c=1.2;$ $t=-2;\ 3;\ 5.67$
6	$y=\begin{cases}at^2\ln t, & 1\le t\le 2\\ 1, & t<1\\ e^{at}\cos bt, & t>2\end{cases}$	$a=-0.5$ $b=2$ $t=1.5;\ 0.5;\ 2.3$
7	$\omega=\begin{cases}x\sqrt[3]{x-a}, & x>a\\ x\sin ax, & x=a\\ e^{-ax}\cos ax, & x<a\end{cases}$	$a=2.5$ $x=3;\ 2.5;\ 1$

Table 2.2 (continuation)

1	2	3
8	$y=\begin{cases}\pi x^2-7/x^2, & x<1.3\\ ax^3+7\sqrt{x}, & x=1.3\\ \lg(x)+7\sqrt{x}, & x>1.3\end{cases}$	$a=1.5$ $x=1;\ 1.3;\ 2$
9	$y=\begin{cases}x, & x<1\\ x^3+6, & 1\le x<2\\ x^2+2, & x=2\\ 0, & x>2\end{cases}$	$x=0.5;\ 1.2;\ 2;\ 4$
10	$g=\begin{cases}ax^3+a\cos^2\lvert x\rvert, & x>0\\ ax^2+bx, & x=0\\ b, & x<0 \text{ and } \lvert x\rvert>2\\ ax+b, & x<0\end{cases}$	$a=2;\quad b=1$ $x=2;\ 0;\ -3.5;\ -1.5$

Questions to Chapter 2.

1. What is the standard expression for `if` statement?
2. When is the operator `switch` used?
3. How should multiple statements be listed?
4. When should the branch structures in the program be used?
5. How should the user run the program and find the errors?

Chapter 3. Programs with iterations

Objectives:

1. For console applications
   To study the iteration statements: *while-statement*, *do-statement* and *for-statement*.
2. For Windows applications
   To study the Common Controls elements: Label, Button, ListBox.

Task 3.1. Calculate the value of the expression.

To write the computer program, which will provide the table with the values of function $y = -2.4x^2 + 5x - 3$ in the range from $-2 \ldots 2$ with increment equal to 0.5. The table should consist of two columns: the column with the argument x values and the column with the corresponding values of function $y(x)$. For the task solution use the iteration statements: *for*, *while*, *do*.

Solution 1.

It creates the console application with *for-statement*. The flowchart of the solution is presented in Fig. 3.1.

The following C# source code solves Task 3.1:

```
using System;
namespace Task3_1_FOR
{
    class Program
    {
        static void Main(string[] args)
        {
            double x, y; /* variables: x for the argument and y
                            for value of function */
            double xs = -2.0;    // start value of x variable
            double xend = 2.0;   // end value of x e
            double dx = 0.5;     // the increment
```

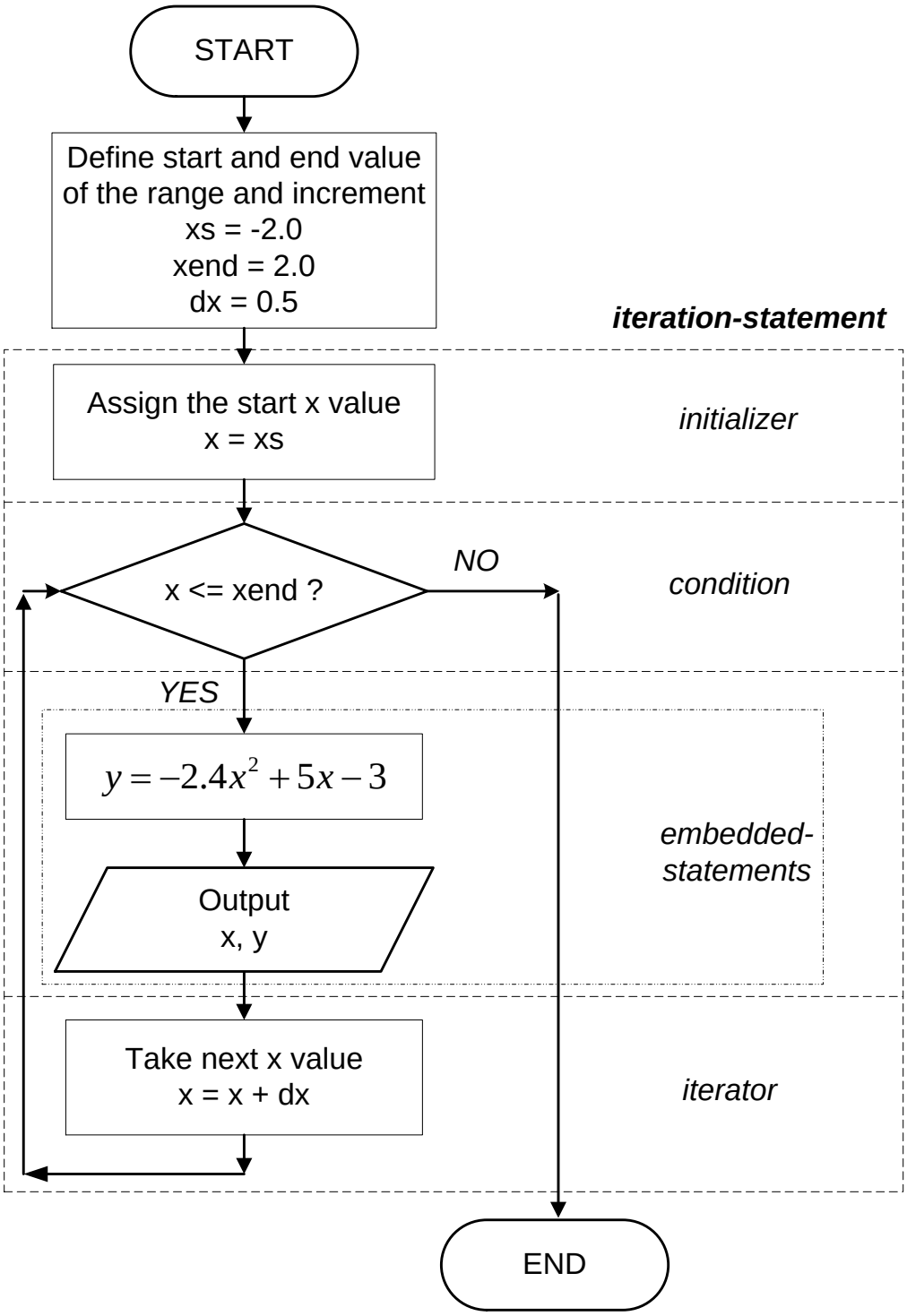


Figure 3.1 - Flowchart of solution for Task 3.1, Solution 1 (*for-statement*)

```
// creating the table in console
 Console.WriteLine("--------------------");
 Console.WriteLine("    x         y     ");
 Console.WriteLine("--------------------");

// beginning the iterations
```

```
            for (x = xs; x <= xend; x = x + dx)
            {
                y = -2.4 * x * x + 5 * x - 3;
                Console.WriteLine("{0,6:F2}\t  {1,6:F3}",x,y);
            }
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 3.2.

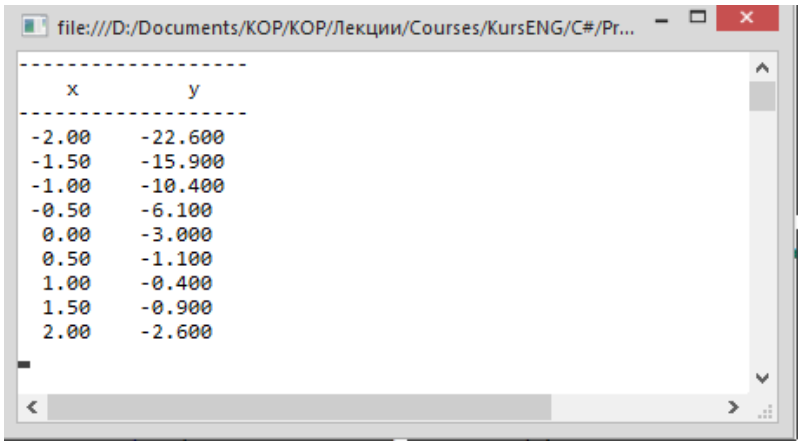


Figure 3.2 - The results of calculations Task 3.1, Solution 1

Note: to stop the endless loop press <**Esc**> or <**Ctrl**> + <**Break**> buttons.

The explanation of the C# code, which is used in the example (Task 3.1, Solution 1):

1) `Console.WriteLine("{0,6:F2}\t  {1,6:F3}",x,y);`

The string `"{0,6:F2}\t  {1,6:F3}"` contains the horizontal tabulation `\t` to insert the equal space between numbers. It helps to display the values in two columns.

2)
```
for (x = xs; x <= xend; x = x + dx)
  {
  }
```

The `for` statement evaluates a sequence of initialization expressions and then, while a condition is true, repeatedly executes an embedded statement and evaluates a sequence of iteration expressions.

for-statement:
`for` (*for-initializer*$_{opt}$; *for-condition*$_{opt}$; *for-iterator*$_{opt}$)
{
embedded-statements
}

The *for-initializer*, if present, consists of either a *local-variable-declaration* or a list of *statement-expression*s separated by commas. The scope of a local variable declared by a *for-initializer* starts at the *local-variable-declarator* for the variable and extends to the end of the embedded statement. The scope includes the *for-condition* and the *for-iterator*.

for-initializer:
local-variable-declaration
statement-expression-list

The *for-condition,* if present, must be a *boolean-expression*.

for-condition:
boolean-expression

The *for-iterator,* if present, consists of a list of *statement-expression*s separated by commas.

for-iterator:
statement-expression-list

A for statement is executed as follows:

- If a *for-initializer* is present, the variable initializers or statement expressions are executed in the order they are written. This step is only performed once.
- If a *for-condition* is present, it is evaluated.
- If the *for-condition* is not present or if the evaluation yields true, control is transferred to the embedded statement. When and if control reaches the end point of the embedded statement (possibly from execution of a continue statement), the expressions of the *for-iterator*, if any, are evaluated in sequence, and then another iteration is performed, starting with evaluation of the *for-condition* in the step above.
- If the *for-condition* is present and the evaluation yields false, control is transferred to the end point of the for statement.

statement-expression-list:
statement-expression
statement-expression-list , *statement-expression*

Within the **embedded statement** of a for statement, a **break** statement may be used to transfer control to the end point of the for statement (thus ending iteration of the embedded statement), and a **continue** statement may be used to transfer control to the end point of the embedded statement (thus executing the *for-iterator* and performing another iteration of the for statement, starting with the *for-condition*).

The embedded statement of a `for` statement is reachable if one of the following is true:

- The `for` statement is reachable and no *for-condition* is present.
- The `for` statement is reachable and a *for-condition* is present and does not have the constant value false.

The end point of a `for` statement is reachable if at least one of the following is true:

- The `for` statement contains a reachable break statement that exits the for statement.
- The `for` statement is reachable and a *for-condition* is present and does not have the constant value true.

In the example:

for-initializer:

```
x = xs
```

The initializer is the assignment to `x` variable the value of start point (`xs`) of the given range.

for-condition:

```
x <= xend
```

The logical statement is to check if the variable `x` is less or equal to the end point (`xend`) of the given range. If it is *true*, the embedded statements run, if *false* then the *for-statement* is reachable, and program ends the iterations and goes to the statement following after the *for-statement*: `Console.ReadLine()`.

for-iterator:

```
x = x + dx
```

The provided *statement-expression* changes the `x` value. Each iteration `x` will be assigned to the increased value of `x` from previous iteration on increment `dx.`

Solution 2.

It creates the console application with *while-statement*. The flowchart of the solution is the same as in Solution1 and presented in Fig. 3.1.

The following C# source code solves Task 3.1:

```
using System;
namespace Task3_1_FOR
{
    class Program
    {
        static void Main(string[] args)
        {
            double x, y; // variables: x for the
            argument and y for value of function
            double xs = -2.0;    // start value of x variable
            double xend = 2.0;   // end value of x e
            double dx = 0.5;     // the increment
            Console.WriteLine("--------------------");
            Console.WriteLine("    x         y    ");
            Console.WriteLine("--------------------");
            x = xs;
            while (x <= xend)
            {
                y = -2.4 * x * x + 5.0 * x - 3.0;
                Console.WriteLine("{0,6:F2}\t  {1,6:F3}",x, y);
                x = x + dx;
            }
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 3.2.

The C# code used in the example (Task 3.1, Solution 2):

```
1) while (x <= xend)  {   }
```

The *while-statement* conditionally executes an embedded statement zero or more times.

***while-statement**:*
while (*boolean-expression*) *embedded-statement*

A while statement is executed as follows:

- The *boolean-expression* (like *for-condition*) is evaluated.
- If the boolean expression yields *true*, control is transferred to the embedded statement. When and if control reaches the end point of the embedded statement (possibly from execution of a continue statement), control is transferred to the beginning of the while statement.
- If the boolean expression yields *false*, control is transferred to the end point of the while statement.

Within the embedded statement of a `while` statement, a break statement may be used to transfer control to the end point of the `while` statement (thus ending iteration of the embedded statement), and a continue statement may be used to transfer control to the end point of the embedded statement.

The *embedded statement* of a `while` statement is reachable if the while statement is reachable and the boolean expression does not have the constant value false.

The end point of a `while` statement is reachable if at least one of the following is true:

- The `while` statement contains a reachable break statement that exits the while statement.
- The `while` statement is reachable and the boolean expression does not have the constant value true.

Solution 3.

It creates the console application with *do-statement*. The flowchart of the solution is presented in Fig. 3.3.

The following C# source code solves Task 3.1:

```
using System;
namespace Task3_1_FOR
{
    class Program
```

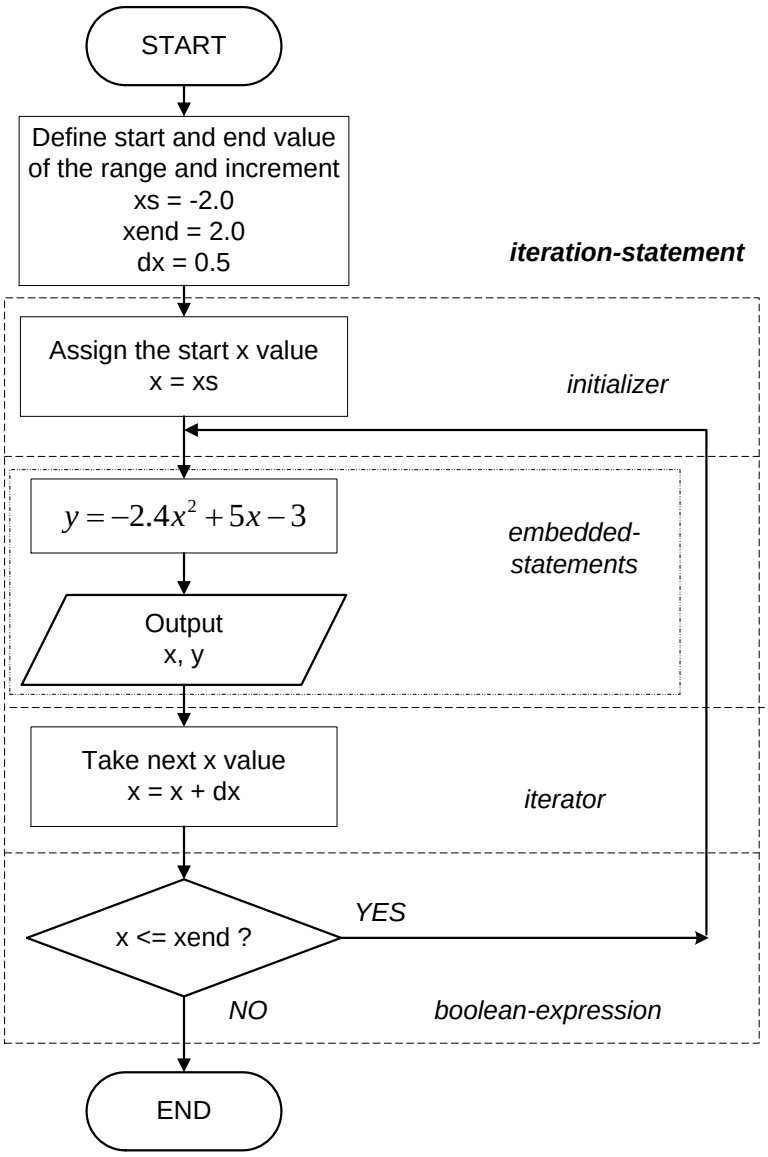


Figure 3.3 - Flowchart of solution for Task 3.1, Solution 3 (*do-statement*)

```
{
    static void Main(string[] args)
    {
        double x, y;  // variables: x for the argument and
                         y for value of function
        double xs = -2.0;    // start value of x variable
        double xend = 2.0;   // end value of x e
        double dx = 0.5;     // the increment
        Console.WriteLine("--------------------");
        Console.WriteLine("    x          y    ");
        Console.WriteLine("--------------------");
        x = xs;
```

```
            do
            {
                y = -2.4 * x * x + 5 * x - 3;
                Console.WriteLine("{0,6:F2}\t  {1,6:F3}",x,y);
                x = x + dx;
            }
            while (x <= xend);
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Figure 3.2.

The C# code used in the example (Task 3.1, Solution 3):

```
1) do {   } while (x <= xend)
```

The *do-statement* conditionally executes an embedded statement one or more times.

***do-statement**:*

do *embedded-statement* while (*boolean-expression*) ;

A *do-statement* is executed as follows:

- Control is transferred to the embedded statement.
- When and if control reaches the end point of the embedded statement (possibly from execution of a continue statement), the *boolean-expression* is evaluated. If the boolean expression yields *true*, control is transferred to the beginning of the do statement. Otherwise, control is transferred to the end point of the do statement.

Within the embedded statement of a *do-statement*, a break statement may be used to transfer control to the end point of the *do-statement* (thus ending iteration of the embedded statement), and a continue statement may be used to transfer control to the end point of the embedded statement.

The embedded statement of a *do-statement* is reachable if the *do-statement* is reachable.

The end point of a *do-statement* is reachable if at least one of the following is true:

- The *do-statement* contains a reachable break statement that exits the *do-statement*.
- The end point of the embedded statement is reachable and the boolean expression does not have the constant value true.

Solution 4.

It creates the Windows Application with *for-statement*.

Do the steps 1 - 5 from Task 1.1, solution 4 to create *Windows application*. The file name – *Task3_1_Solution4* in ***D:/Documents/O65/Programs*** folder.

6. Open the *Common controls* tab in **Toolbox** window and drag the following objects to the *Form 1*: **Label**; **ListBox**; **Button**. Allocate them in *Form 1* according to Fig. 3.4.

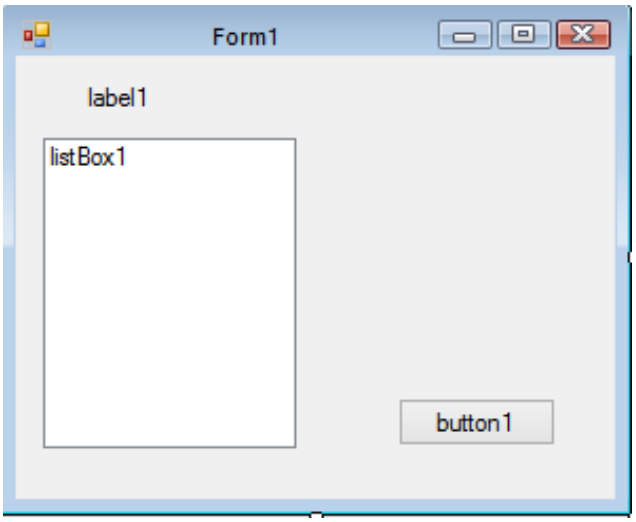


Figure 3.4 - Allocation of the objects used in *Form 1*

7. Change the properties of these objects in **Properties** Window as listed in Table 3.1.

Table 3.1 - The properties of the objects from *Form 1*.

Object	The default name of the object (value of Name property field)	Property	New value
Form	Form1		
Label	label1	Text	Calculation results
ListBox	listBox1		
Button	button1	Name	cmdStart
		Text	Calculate
		Font	Times New Roman, Bold, 12

The *Form 1* will be modified according to Fig. 3.5.

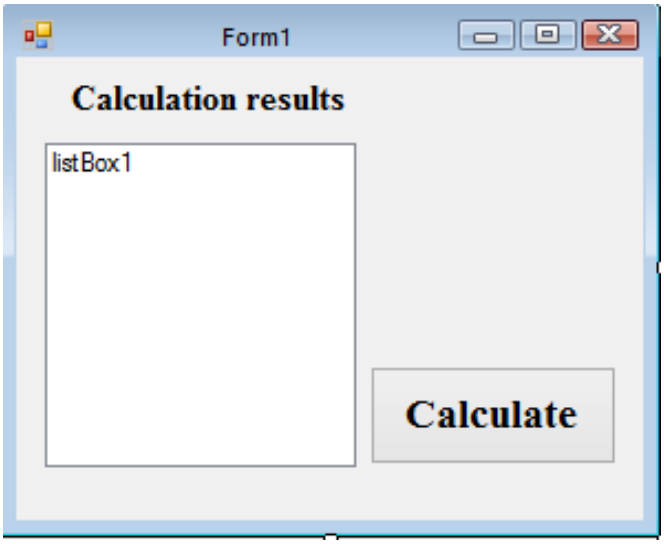


Figure 3.5 - Objects in *Form 1* after changing their properties

8. Write the C# code for solving the Task 3.1, working when the created **Calculate** button is pressed.

```
using System;
using System.Windows.Forms;
namespace Task3_1_Solution4
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void cmdStart_Click(object sender, EventArgs e)
        {
            double x, y;          /* variables: x for the
                argument and y for value of function */
            double xs = -2.0;     // start value of x variable
            double xend = 2.0;    // end value of x
            double dx = 0.5;      // the increment
            listBox1.Items.Add("-------------------------");
            listBox1.Items.Add("    x               y    ");
            listBox1.Items.Add("-------------------------");
            for (x = xs; x <= xend; x = x + dx)
```

```
            {
                y = -2.4 * x * x + 5 * x - 3;
                listBox1.Items.Add(x.ToString("F2") + "\t" +
                                   y.ToString("F3"));
            }
        }
    }
}
```

9. Run the program.
Press Start button (Fig. 1.6) or F5.
It will give the results of calculations presented in Fig. 3.6.

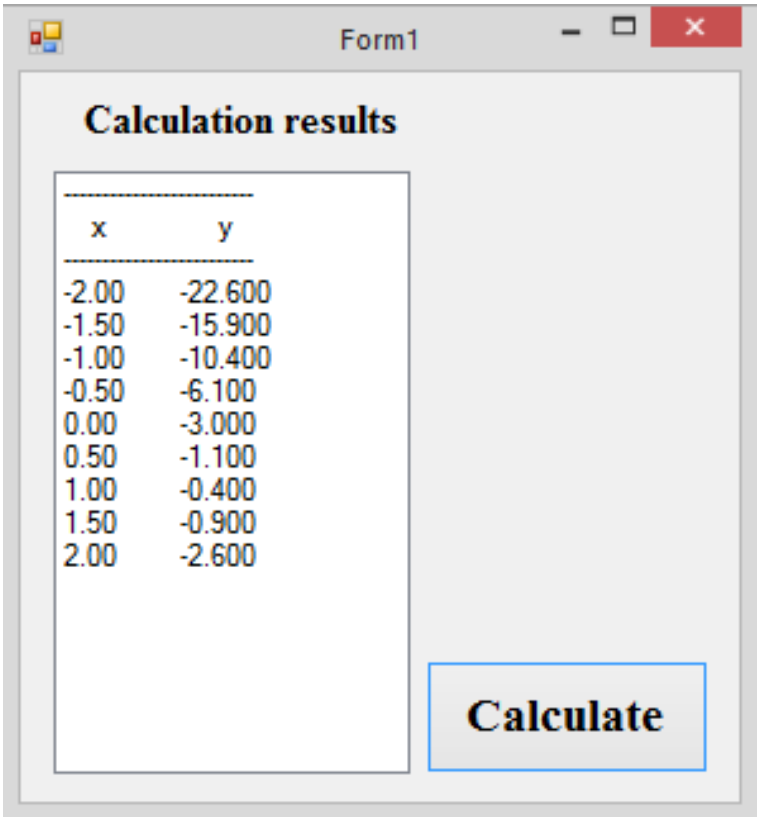


Figure 3.6 - Window with the results of calculations

10. Close the window with the results or go to DEBUG → **Stop Debugging** (to stop the program debugging process).

11. Save your Project by clicking FILE → **Save all**.

Task 3.2.

To write the console application, which will calculate the value of factorial $k!$, where $k! = 1 \cdot 2 \cdot 3 \cdot ... \cdot (k-1) \cdot k$. The flowchart of the solution is presented in Fig. 3.7.

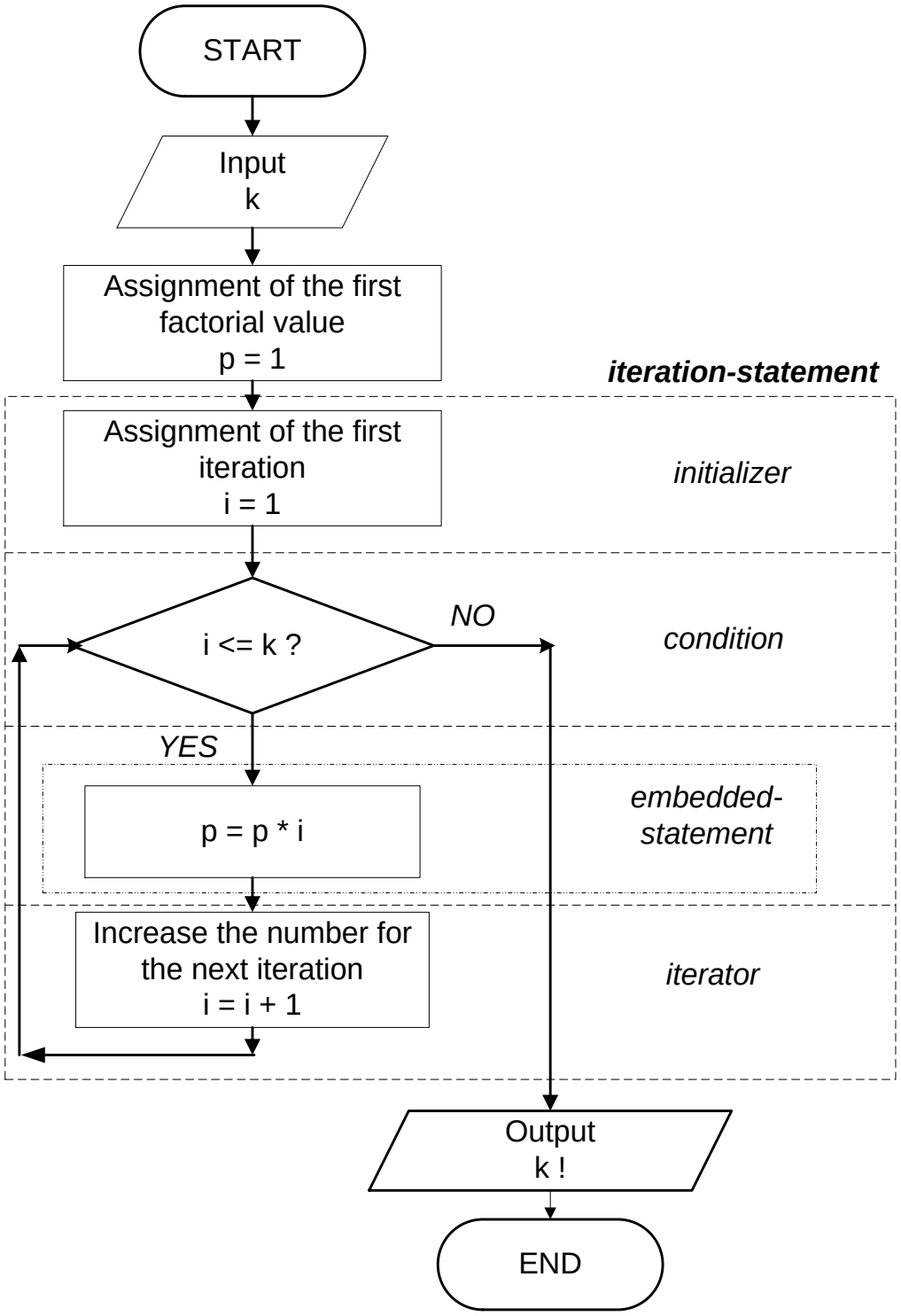


Figure 3.7 - Flowchart of solution for Task 3.2

The following C# source code solves Task 3.2:

```
using System;
namespace Task3_2
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, k;
/* declaration of integer values: i for iterations; k for the
input number */
            double p;
// the number to assign the results of calculations
            Console.Write(" Input k= ");
            k = Convert.ToInt32(Console.ReadLine());
// converts the string variable entered by user to integer type
            p = 1;
// assigns the initial value for the factorial calculation
            for (i = 1; i <= k; i++)
                p = p * i;  /* each iteration the previous p
value is multiplyed by the iteration number */
            Console.WriteLine("{0}! = {1}", k, p);
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 3.8.

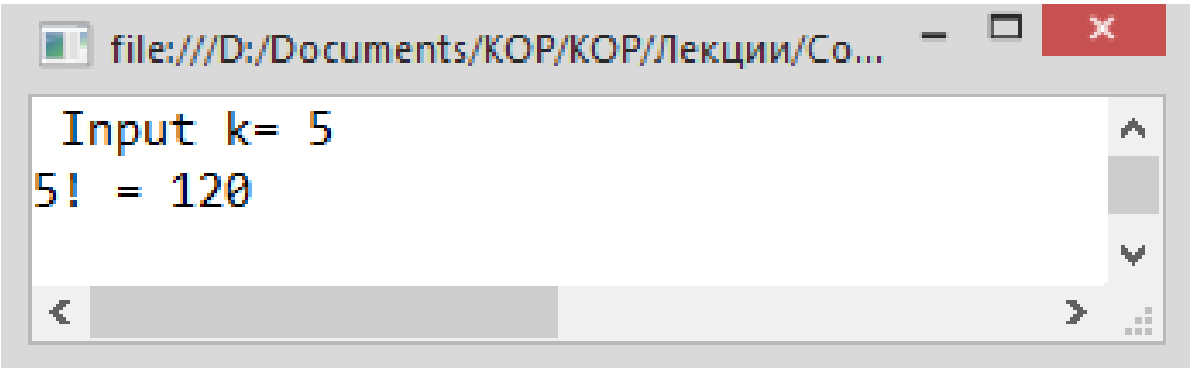


Figure 3.8 - The results of calculations Task 3.2

Task 3.3.

To write the console application, which will calculate the value of first n terms of the series

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{n}\ .$$

The number n is defined by the user from keyboard in the following form:

Input:	Input the number n of summaries terms of the series n → ____
Output:	The summation of first ___ terms of the series is equal _____

The following C# source code solves Task 3.3:

```
using System;
namespace Task3_3
{
    class Program
    {
        static void Main(string[] args)
        {
          int n;        // the number of terms in series
          int i;        // the number of the current term
          double elem;  // the calculated value of the term
          double sum;   // the summation of the terms in series
          Console.Write("Input the number n of summaries terms
of the series n->");
          n = Convert.ToInt32(Console.ReadLine());
          Console.WriteLine(" n= " + n);
          sum = 0;
          for (i = 1; i <= n; i = i + 1)
            {
                elem = 1.0 / i;
                sum = sum + elem;
            }
          Console.WriteLine();
          Console.WriteLine("The summation of first {0} terms
of the series is equal  =   {1,6:F4} ", n, sum);
```

```
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 3.9.

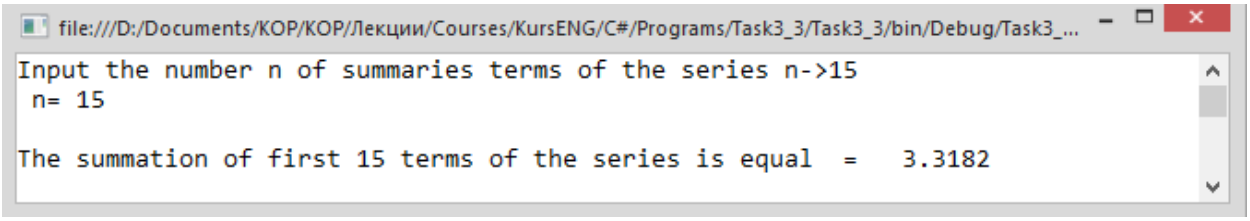


Figure 3.9 - The results of calculations Task 3.3

Task 3.4.

To output the values of the following function depending from x:

$$y(x)=\begin{cases} t, & x<0 \\ t\cdot x & \text{if } 0\le x<10, \\ 2t & x\ge 10 \end{cases}$$

where $t=5$; x varies in the range $-2\le x\le 12$, increment is equal 2.

The program should provide the table with two columns: one with x values and the second with corresponding $y(x)$ values.

The following C# source code solves Task 3.4:

```
using System;
namespace Task3_4
{
    class Program
    {
        static void Main(string[] args)
        {
            double x, y, t = 5;
            double xs = -2;
// the start point of x in the range
            double xe = 12;
// the end point of x in the range
```

```
            double dx = 2;        // the increment
            //  The initial rows for the table
            Console.WriteLine("------------------");
            Console.WriteLine("   x         y    ");
            Console.WriteLine("------------------");
            // the calculation of values and their output
            for (x = xs; x <= xe; x = x + dx)
            {
                if (x < 0)
                    y = t;
                else if (0 <= x && x < 10)
                    y = t * x;
                else
                    y = 2 * t;
                Console.WriteLine("{0,6:F2}\t  {1,6:F3}",x, y);
            }
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 3.10.

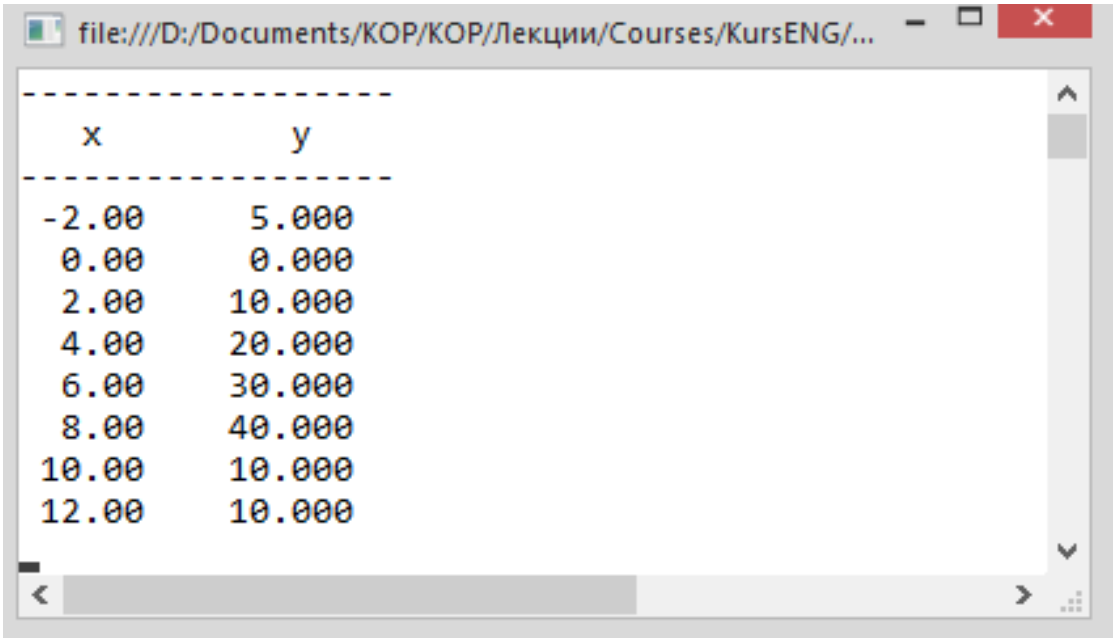


Figure 3.10 - The results of calculations Task 3.4

Task 3.5. Tasks for laboratory works.

To write the program providing the table of two columns: one with the argument values and the second with corresponding values of the function. The values of argument, the range of its variation, function and used constants are listed in Table 3.2.

Table 3.2 - Variants for calculation

<table>
<tr><th>№</th><th>Function</th><th>The range of argument variation</th><th>Increment</th><th>Constants</th></tr>
<tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr>
<tr><td>1</td><td>$y(x) = x + bx\ Sin\ L$</td><td>$0 \le x \le 1$</td><td>$\Delta = 0.1$</td><td>$b = 2.3$;
$L = 1.45$</td></tr>
<tr><td>2</td><td>$y(x) = x + 3x^2 - 2,5x^3 + 0,75x^4$</td><td>$1 \le x \le 4$</td><td>$\Delta = 0.5$</td><td></td></tr>
<tr><td>3</td><td>$y(m) = 8m + 4m^4 - 1,5m^3$</td><td>$1 \le m \le 3$</td><td>$\Delta = 0.2$</td><td></td></tr>
<tr><td>4</td><td>$S1(r) = 4\pi r^2;\quad S2(r) = \pi r^2/2$
The table should be of the following form:
<table>
<tr><th>R</th><th>S1</th><th>S2</th></tr>
<tr><td>1.00</td><td>12.566</td><td>1.571</td></tr>
<tr><td>.</td><td>.</td><td>.</td></tr>
<tr><td>.</td><td>.</td><td>.</td></tr>
<tr><td>.</td><td>.</td><td>.</td></tr>
<tr><td>.</td><td>.</td><td>.</td></tr>
<tr><td>1.40</td><td>24.630</td><td>3.079</td></tr>
</table></td><td>$1 \le r \le 1.4$</td><td>$\Delta = 0.05$</td><td></td></tr>
<tr><td>5</td><td>$y(x) = \lvert x \rvert$</td><td>$-4 \le x \le 4$</td><td>$\Delta = 0.5$</td><td></td></tr>
<tr><td>6</td><td>$y(x) = \lvert x - 2 \rvert + \lvert x + 1 \rvert$</td><td>$-4 \le x \le 4$</td><td>$\Delta = 0.5$</td><td></td></tr>
<tr><td>7</td><td>$y(x) = 2x^3 - 15x^2 + 4,7x - 3$</td><td>$-3 \le x \le 3$</td><td>$\Delta = 0.5$</td><td></td></tr>
<tr><td>8</td><td>$y(x) = 5^{\cos(x)} + 5\sin(x)^2 - 3$</td><td>$1 \le x \le 5$</td><td>$\Delta = 0.5$</td><td></td></tr>
<tr><td>9</td><td>$y(x) = \dfrac{3\sin(x)^3 + 2.3}{x - 2\cos(x)^3}$</td><td>$3 \le x \le 9$</td><td>$\Delta = 0.5$</td><td></td></tr>
<tr><td>10</td><td>$y(x) = \left(\sin(x)^3 - 0.6\right)/x^2$</td><td>$1 \le x \le 10$</td><td>$\Delta = 1$</td><td></td></tr>
</table>

Questions to Chapter 3.

1. Describe the standard *while-statement* expression.
2. Describe the standard *do-statement* expression.
3. Describe the standard *for-statement* expression.
4. When should **break** and **continue** statements be used?
5. What is the main difference between using *while-statement, do-statement* and *for-statement*?

Chapter 4. Vectors or one-dimension arrays

Objectives:

3. For console applications
   To study the operators of working with vectors: the declaration of vector, ways of its filling in, input and output of vector elements, filtering of elements of vector, finding the maximal element in vector.
4. For Windows applications
   To study the Common Controls elements: Label, Button, TextBox, ListBox, ComboBox.

Task 4.1.

Calculate the following expression: $z = \sum_{i=4}^{10} y_i$

where

y_i are the elements of vector $Y(12)$, $i = 0,1, \ldots 11$.

$\sum_{i=4}^{10} y_i$ is the sum of Y vector elements from 4^{th} to 10^{th}.

Here the notation $Y(12)$ means the vector $Y = (y_i;\ i = 0,1,\ldots 11)$, the vector size (the number of its elements) is equal to 12. The number of the starting element is = 0, the number of the end element is equal = 11.

Solution 1.

It creates the console application and vector elements are assigned in the C# code. The vector size is determined by the constant.

The following C# source code solves the Task 4.1:

```
using System;
namespace Task3_4_Solution1
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
            const int n = 12;  // the number of vector elements
            int i; double z, s;
            double[] Y = { 1.0, 2.5,3.7,4.8,5.0,6.2, 7.3, 8.0,
                                        9.5, 10.0, 12.0, 14.0 };
            Console.WriteLine("The vector elements are: ");
            // The output of vector elements
            for (i = 0; i <= n - 1; i = i + 1)
                Console.WriteLine(" " + Y[i]);
        // Calculation of sum of the elements from 4th to 10th
            s = 0;
            for (i = 4; i <= 10; i = i + 1)
                s = s + Y[i];
            z = s;
            Console.WriteLine();
            Console.WriteLine(
                      "Sum of the elements is equal= {0}", z);
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 4.1.

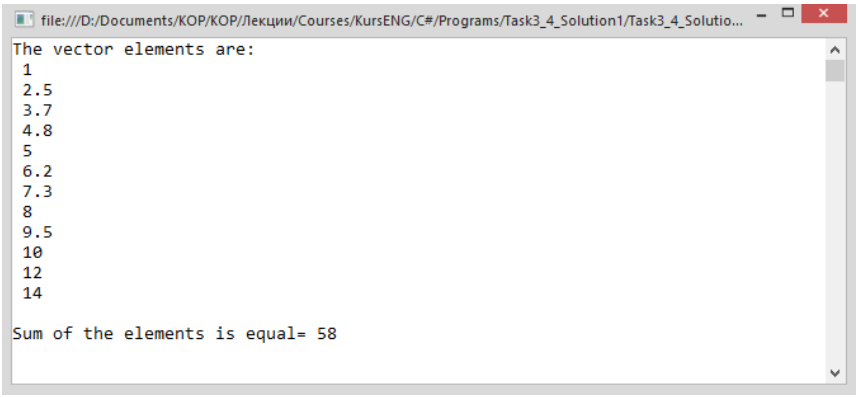


Figure 4.1 - The results of calculations Task 4.1, Solution 1

The C# code used in Task 4.1 (Solution 1):

```
1) double[] Y = { 1.0, 2.5,3.7,4.8,5.0,6.2, 7.3, 8.0,
                                        9.5, 10.0, 12.0, 14.0 };
```

Array elements

The vector is the one-dimensional array and all information in C# about arrays is applied to vectors. The elements of an array come into existence when an array instance is created, and cease to exist when there are no references to that array instance. The initial value of each of the elements of an array is the default value of the type of the array elements. For the purpose of definite assignment checking, an array element is considered initially assigned.

```
2) Console.WriteLine(" " + Y[i])
```

Element access

An *element-access* consists of a *primary-no-array-creation-expression*, followed by a "[" token, followed by an *argument-list*, followed by a "]" token. The *argument-list* consists of one or more *argument*s, separated by commas.

> *element-access:*
> *primary-no-array-creation-expression* [*argument-list*]

For an array access, the *primary-no-array-creation-expression* of the *element-access* must be a value of an *array-type*. The *argument-list* of an array access is not allowed to contain named arguments. The number of expressions in the *argument-list* must be the same as the rank of the *array-type*, and each expression must be of type int, uint, long, ulong, or must be implicitly convertible to one or more of these types.

The result of evaluating an array access is a variable of the element type of the array, namely the array element selected by the value(s) of the expression(s) in the *argument-list*.

The run-time processing of an array access of the form P[A], where P is a *primary-no-array-creation-expression* of an *array-type* and A is an *argument-list*, consists of the following steps:

- P is evaluated. If this evaluation causes an exception, no further steps are executed.
- The index expressions of the *argument-list* are evaluated in order, from left to right. Following evaluation of each index expression, an implicit conversion to one of the following types is performed: int, uint, long, ulong. The first type in this list for which an implicit conversion exists is chosen. If evaluation of an index expression or the subsequent implicit conversion causes an exception, then no further index expressions are evaluated and no further steps are executed.

– The value of P is checked to be valid. If the value of P is null, a System.NullReferenceException is thrown and no further steps are executed.
– The value of each expression in the *argument-list* is checked against the actual bounds of each dimension of the array instance referenced by P. If one or more values are out of range, a System.IndexOutOfRangeException is thrown and no further steps are executed.
– The location of the array element given by the index expression(s) is computed, and this location becomes the result of the array access.

Here Y[*i*] outputs *i* element of Y vector with **Console.WriteLine** method. The elements are displayed consequently using iterations (*for-statement*).

Solution 2.

It creates the console application and vector elements are calculated according to the formula: $y_i = i/2.0 + 7$. The vector size is determined by the constant.

The following C# source code solves the Task 4.1:

```
using System;
namespace Task4_1_Solution2
{
    class Program
    {
        static void Main(string[] args)
        {
            const int n = 12;
                        // The number of vector elements
            double[] Y = new double[n];
            int i; double z, s;
            Console.WriteLine("Vector elements are: ");
            Console.WriteLine();
            for (i = 0; i <= n - 1; i = i + 1)
            {
                Y[i] = i / 2.0 + 7;
                Console.WriteLine("Y[{0}]={1} ", i, Y[i]);
            }
```

```
            s = 0;
            for (i = 4; i <= 10; i = i + 1)
                s = s + Y[i];
            z = s;
            Console.WriteLine();
            Console.WriteLine(
                "Sum of the elements is equal = {0}", z);
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 4.2.

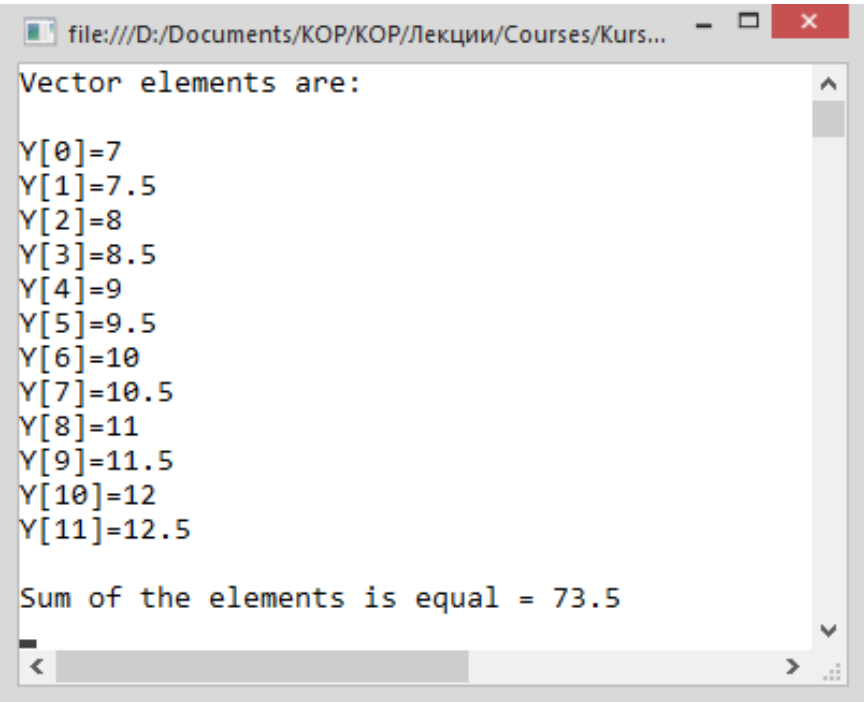


Figure 4.2 - The results of calculations Task 4.1, Solution 2

Solution 3.

It creates the console application and vector elements are input by the user from keyboard. The vector size is determined by the constant.

The following C# source code solves the Task 4.1:

```
using System;
namespace Task4_1_solution3
{
    class Program
    {
        static void Main(string[] args)
        {
            const int n = 12;  // the number of vector elements
            double[] Y = new double[n];
            int i; double z, s;
            Console.WriteLine("Input of the vector: ");
            for (i = 0; i <= n - 1; i = i + 1)
            {
                Console.Write("Input "+i+" vector element = ");
                Y[i] = Convert.ToDouble(Console.ReadLine());
            }
            Console.WriteLine();
            Console.WriteLine("Vector elements are: ");
            for (i = 0; i <= n - 1; i = i + 1)
            {
                Console.WriteLine("Y[{0}]={1} ", i, Y[i]);
            }
            s = 0;
            for (i = 4; i <= 10; i = i + 1)
                s = s + Y[i];
            z = s;
            Console.WriteLine(
                   "Sum of the elements is equal = {0} ", z);
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 4.3.

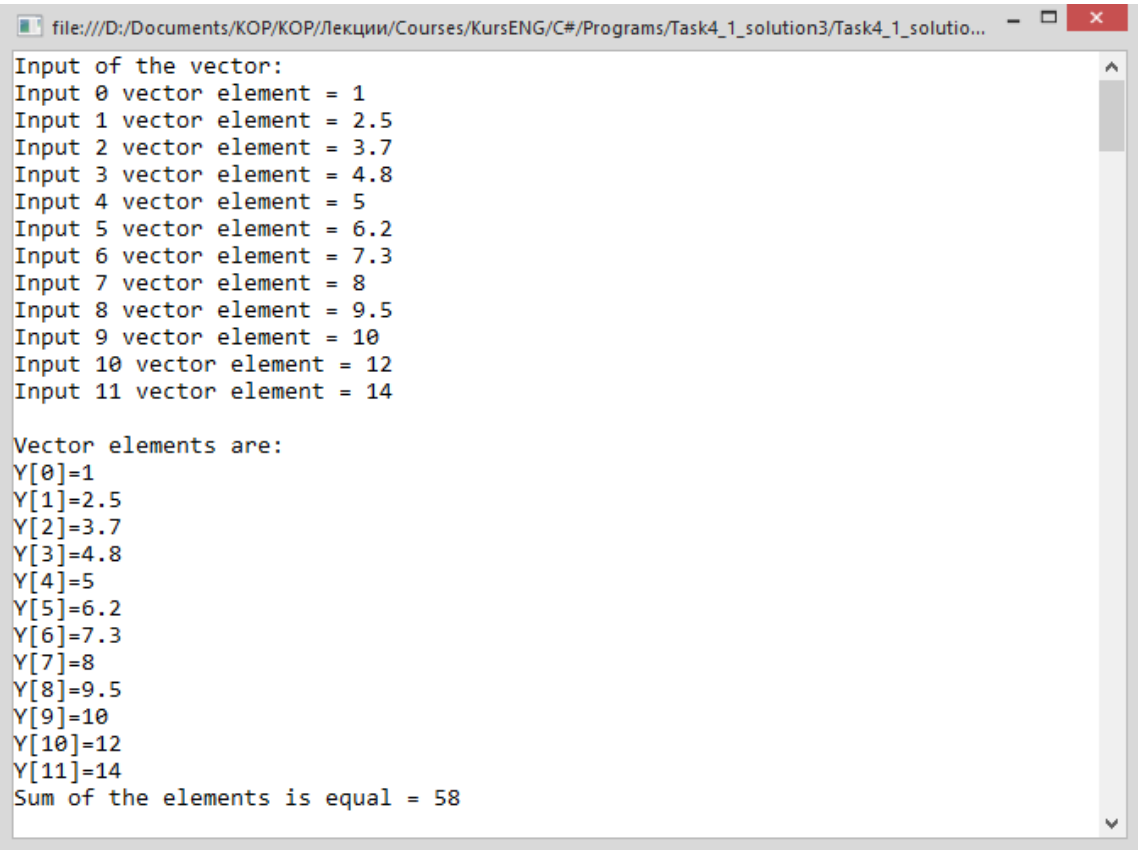


Figure 4.3 - The results of calculations Task 4.1, Solution 3

Solution 4.

It creates the console application and vector elements are determined using the random number generator from the range 0.0 ... 10.0. The vector size is determined by the constant.

The following C# source code solves the Task 4.1:

```
using System;
namespace Task4_1_Solution4
{
    class Program
    {
        static void Main(string[] args)
        {
            const int n = 12; // the number of vector elements
            double[] Y = new double[n];
            int i; double z, s;
            Random rnd = new Random();
            Console.WriteLine("The vector elements are: ");
```

```
            for (i = 0; i <= n - 1; i = i + 1)
            {
                Y[i] = rnd.NextDouble() * 10;
                Console.WriteLine("Y[{0}]={1,5:F2} ", i, Y[i]);
            }
            s = 0;
            for (i = 4; i <= 10; i = i + 1)
                s = s + Y[i];
            z = s;
            Console.WriteLine();
            Console.WriteLine(
              "Sum of the elements is equal = {0,6:F3}", z);
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 4.4.

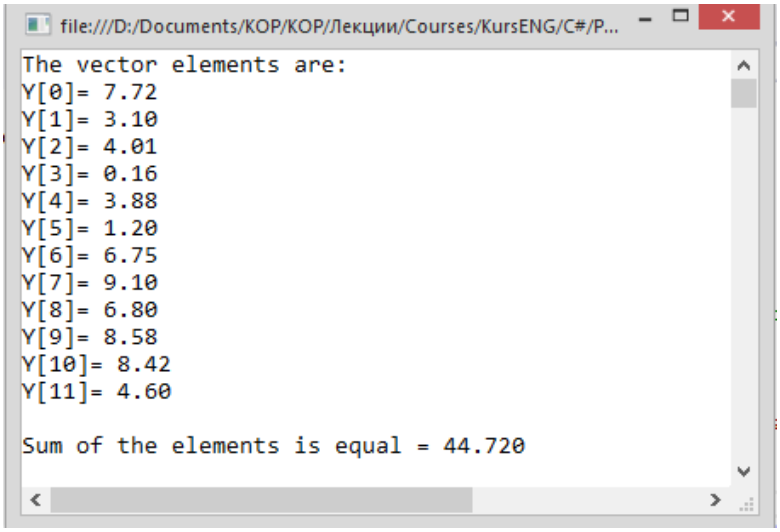


Figure 4.4 - The results of calculations Task 4.1, Solution 4

The C# code used in Task 4.1 (Solution 4):

```
1) double[] Y = new double[n];
   …
   Random rnd = new Random();
      …
```

```
for (i = 0; i <= n - 1; i = i + 1)
{
    Y[i] = rnd.NextDouble() * 10;
    …
}
```

Random Class

Random Class represents a pseudo-random number generator, a device that produces a sequence of numbers that meet certain statistical requirements for randomness.

Random Constructor `Random()` initializes a new instance of the Random class, using a time-dependent default seed value. **Random.NextDouble()** Method Returns a random number between 0.0 and 1.0.

Solution 5.

It creates the Windows Application, in which the vector elements are calculated according to formula in Solution 2, the vector elements are displayed in ListBox. The sum of the elements is displayed in Label.

1. Open the *Common controls* tab in **Toolbox** window and drag the following objects to the *Form 1*: **Label** (2 pieces); **ListBox**; **Button**. Allocate them in *Form 1* according to Fig. 4.5.

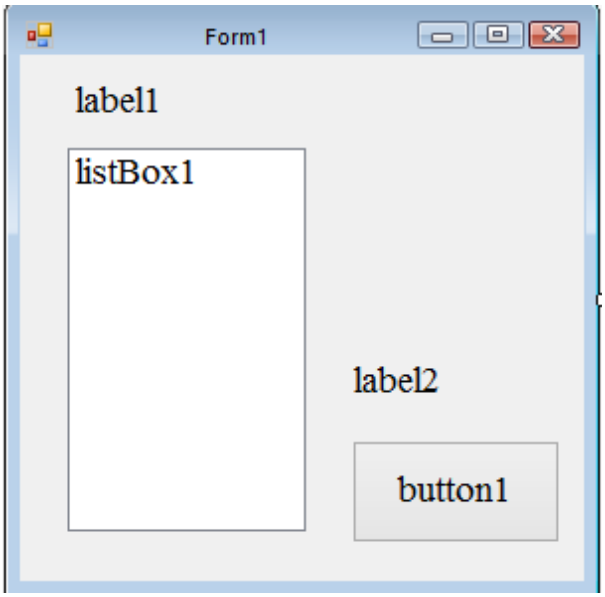


Figure 4.5 - Allocation of the objects used in *Form 1*

2. Change the properties of these objects in **Properties** Window as listed in Table 4.1.

Table 4.1 - The properties of the objects from *Form 1*

Object	The default name of the object (value of Name property field)	Property	New value
Form	Form1		
Label	label1	text	Vector Y elements
Label	label2	text	
ListBox	listBox1	Name	listBox1
Button	button1	Name	cmdStart
		Text	Calculate

The *Form 1* will be modified according to Fig. 4.6.

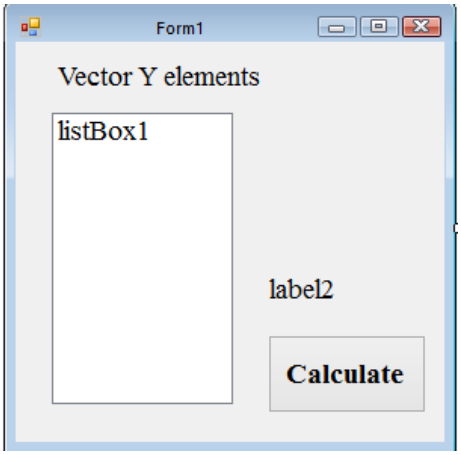


Figure 4.6 - Objects in *Form 1* after changing their properties

3. Write the C# code for solving the Task 3.1, working when pressing the **Calculate** button.

```
using System;
using System.Windows.Forms;
namespace Task4_1_Solution5
{
    public partial class Form1 : Form
    {
        public Form1()
```

```
        {
            InitializeComponent();
        }
        private void cmdStart_Click(object sender, EventArgs e)
        {
            const int n = 12; // the number of vector elements
            double[] Y = new double[n];
            int i; double z, s;
            for (i = 0; i <= n - 1; i = i + 1)
            {
                // Calculation of the vector elements
                Y[i] = i / 2.0 + 7;
                // The output of the vector elements in ListBox
                listBox1.Items.Add(Y[i].ToString("F2"));
            }
            s = 0;
            for (i = 4; i <= 10; i = i + 1)
                s = s + Y[i];
            z = s;
            Console.WriteLine();
            label2.Text = (" Sum of the elements \n
          from 4th to 10th \n is equal =" + z.ToString("F3"));
        }
    }
}
```

It will give the results of calculations presented in Fig. 4.7.

The C# code used in Task 4.1 (Solution 5):

`1) listBox1.Items.Add(Y[i].ToString("F2"))`

ListBox.ObjectCollection Class represents the collection of items in a ListBox.

ListBox.Items Property gets the items of the ListBox. This property enables you to obtain a reference to the list of items that are currently stored in the ListBox. With this reference, you can add items, remove items, and obtain a count of the items in the collection.

ListBox.ObjectCollection.Add Method adds an item to the list of items for a ListBox.

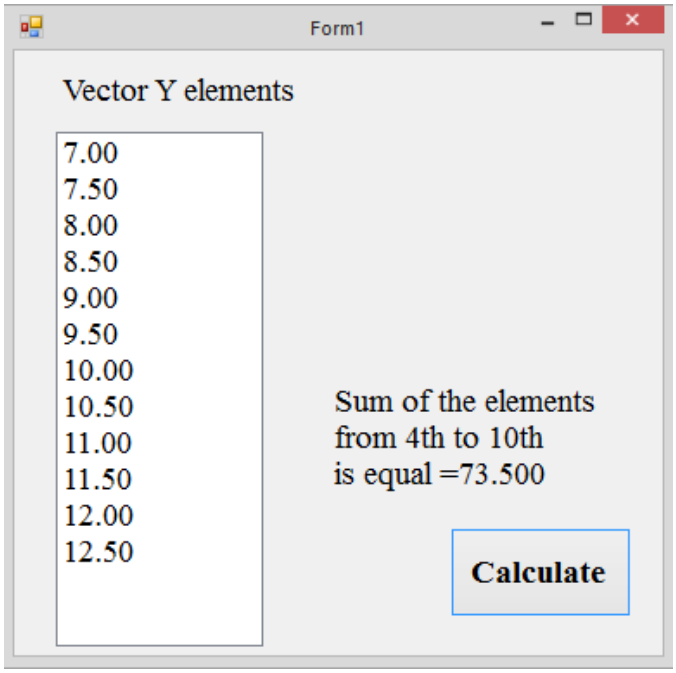


Figure 4.7 - Window with the results of calculations

Task 4.2.

To write the console application for solving the following problem.

Calculate the following expression: $z = \prod_{i=0}^{11} a_i$

where

a_i are the elements of vector $A(12)$, $i = 0,1,\ldots 11$.

$\prod_{i=0}^{11} a_i$ is the product of all elements of vector $A(12)$

The values of vector elements are calculated according to formula $a_i = (i + 3.0)/7.0$.

The following C# source code solves the Task 4.2:

```
using System;
namespace Task4_2
{
    class Program
    {
        static void Main(string[] args)
        {
            const int n = 12;  // the number of vector elements
```

```
            double[] A = new double[n];
            int i; double z, p;
            Console.WriteLine("The vector elements are: ");
            Console.WriteLine();
            for (i = 0; i<=n-1; i = i + 1)
            {
                A[i] = (i+3.0)/7.0;
                Console.WriteLine("A[{0}]={1,5:F2} ", i, A[i]);
            }
            p = 1;
            for (i = 0; i <= n-1; i = i + 1)
                p = p * A[i];
            z = p;
            Console.WriteLine();
            Console.WriteLine("Product of the vector elements
                              is equal = {0,7:F3} ", z);
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 4.8.

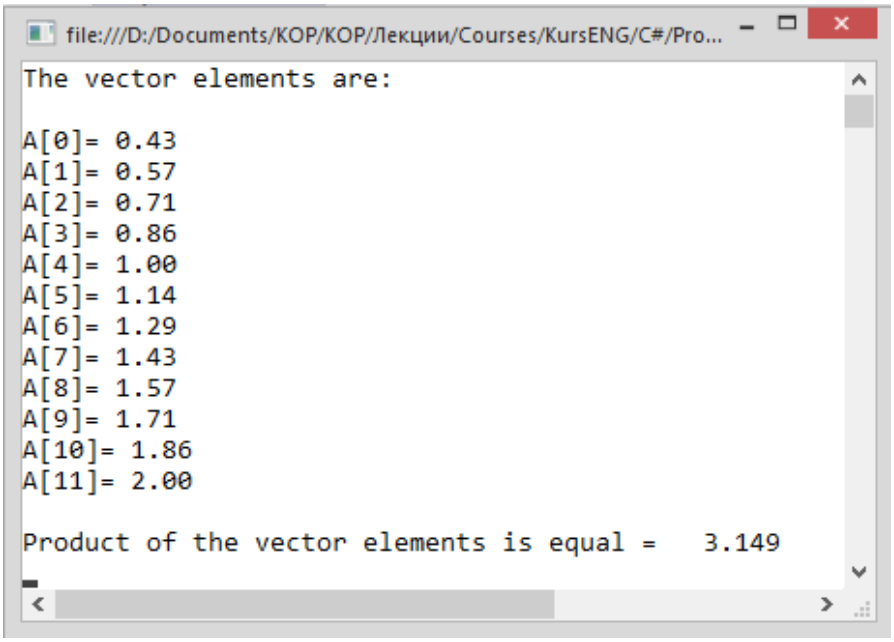


Figure 4.8 - The results of calculations Task 4.2

Task 4.3.

To sort out the elements of vector $A(12)$. The vector elements are assigned by the program. Use the Array.Sort Method to sort the elements in vector.

The following C# source code solves the Task 4.3:

```
using System;
namespace Task4_3
{
    class Program
    {
        static void Main(string[] args)
        {
            const int n = 10; // the number of vector elements
            int i;
            int[] A = { 12, 4, 7, 9, 10, 5, 15, 2, 3, 1 };
            Console.WriteLine("The initial vector is ");
            // The output of the vecot elements
            for (i = 0; i <= n - 1; i = i + 1)
                Console.Write(" " + A[i]);
            Console.WriteLine();
            Array.Sort(A);
            Console.WriteLine();
            Console.WriteLine("The vector sorted out is ");
            // The output of the vector, which was sorted out
            for (i = 0; i <= n - 1; i = i + 1)
                Console.Write(" " + A[i]);
            Console.WriteLine();
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 4.9.

The C# code used in Task 4.3:

`1) Array.Sort(A)`

Array.Sort Method sorts the elements in one-dimensional Array objects. Array.Sort Method (Array) sorts the elements in an entire one-dimensional Array using the IComparable implementation of each element of the Array. Each

element of array must implement the Icomparable interface to be capable of comparisons with every other element in array.

If the sort is not successfully completed, the results are undefined.

This method uses the *QuickSort* algorithm. This implementation performs an unstable sort; that is, if two elements are equal, their order might not be preserved. In contrast, a stable sort preserves the order of elements that are equal.

On average, this method is an O(n log n) operation, where n is the Length of array; in the worst case it is an O(n ^ 2) operation.

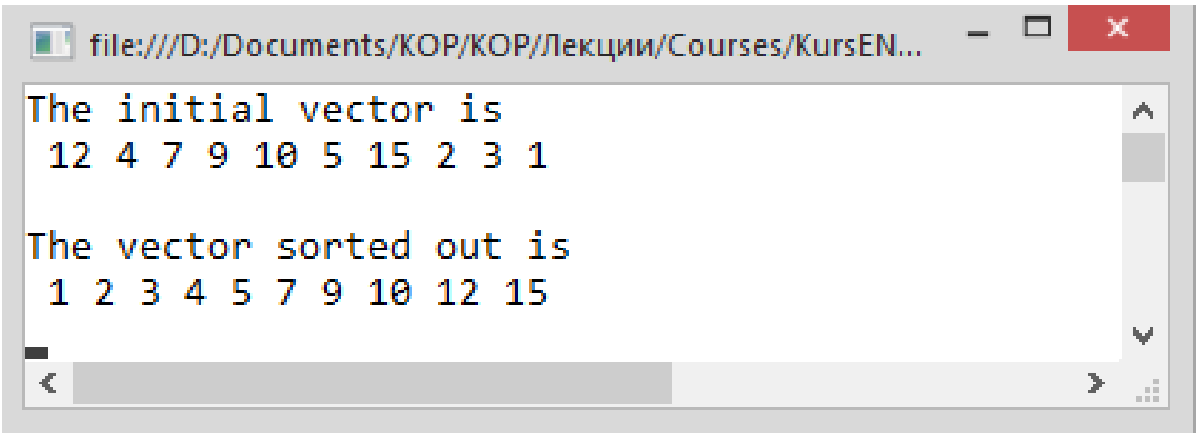


Figure 4.9 - The results of calculations Task 4.3

Task 4.4.

To develop the console application, which will calculate the arithmetical mean number of one-dimensional array $A(n)$, where n is the number of array elements. The n number should be specified by the user. The array element should be obtained using the random number generator.

The following C# source code solves the Task 4.4:

```
using System;
namespace Task4_4
{
    class Program
    {
        static void Main(string[] args)
        {
            int n;      // The number of array elements
            int i;
```

```
            double s;  // Sum of array elements
            double sr;
              // The arithmetic mean number of array elements
            Console.Write("Enter the number of elements
                          in array = ");
            n = Convert.ToInt32(Console.ReadLine());
            double[] A = new double[n];
            Random rnd = new Random();
            Console.WriteLine();
            Console.WriteLine("The array elements are: ");
            s = 0;
            for (i = 0; i <= n-1; i = i + 1)
            {
                A[i] = rnd.Next(1, 20);
                Console.WriteLine("A[{0}]={1} ", i, A[i]);
                s = s + A[i];
            }
            sr = s / n;
            Console.WriteLine();
            Console.WriteLine("The arithmetic mean number of
                       the array is equal = {0,6:F3}", sr);
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 4.10.

Task 4.5.

To develop the console application, which calculate the maximal element of one-dimensional array $A(12)$ with real numbers and provide the number of this element. The array element should be input in program.

The following C# source code solves the Task 4.5:

```
using System;
namespace Task4_5
{
    class Program
```

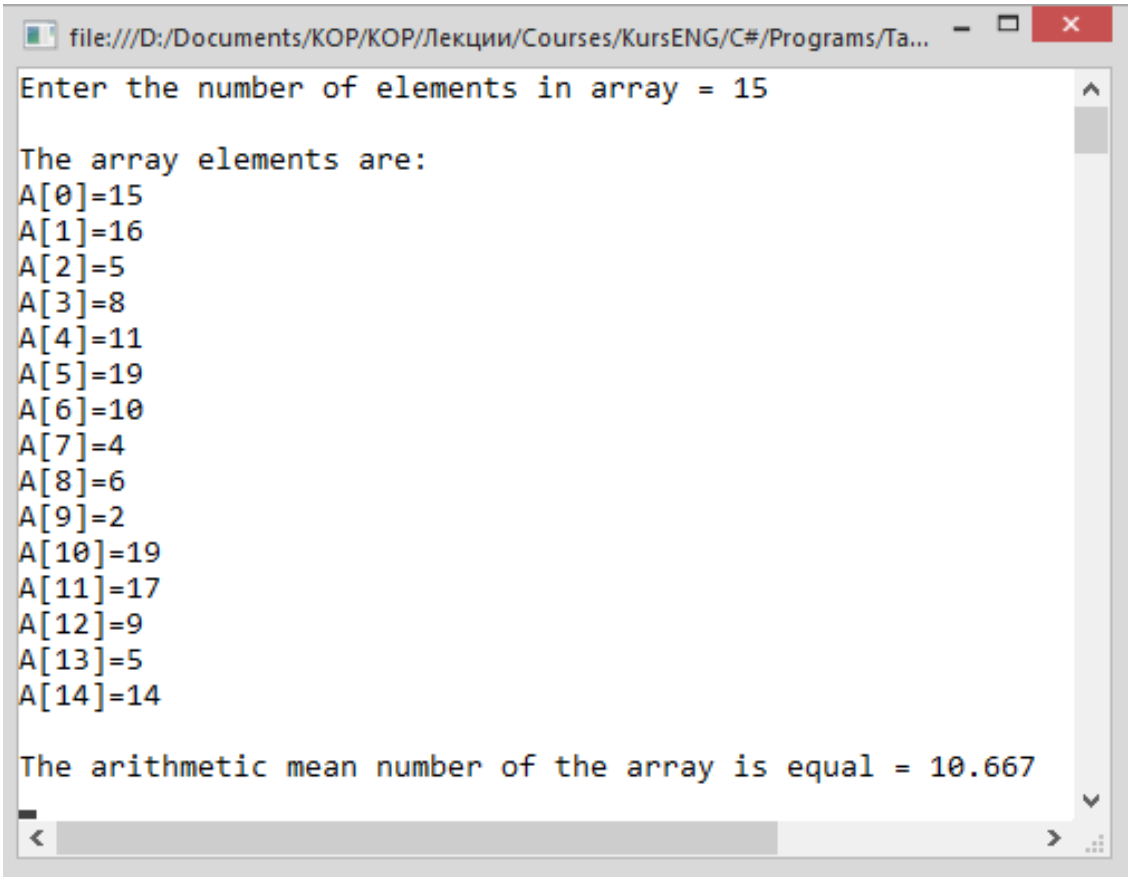


Figure 4.10 - The results of calculations Task 4.4

```
{
    static void Main(string[] args)
    {
        const int n = 12;  // The number of array elements
        double max; // The maximal element
        int imax;   // The index (number) of the maximal element
        int i;
        double[] A = { 1.0, -2.5, 3.7, 94.8, 15.0, 65.2, -
                       7.3, 18.0, 29.5, 10.0, 17.0, 14.0 };
        Console.WriteLine("The array elements are: ");
        Console.WriteLine();
        // The output of array elements
        for (i = 0; i <= (n - 1); i = i + 1)
            Console.WriteLine("A[{0}]={1} ", i, A[i]);
        max = A[0]; imax = 0;
        for (i = 0; i <= (n - 1); i = i + 1)
        {
            if (A[i] > max)
```

```
                {
                    max = A[i];
                    imax = i;
                }
            }
            Console.WriteLine();
            Console.WriteLine("The maximal element of the array
                               is equal = {0}", max);
            Console.WriteLine("The index of the maximal element
                               is = {0}", imax);
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 4.11.

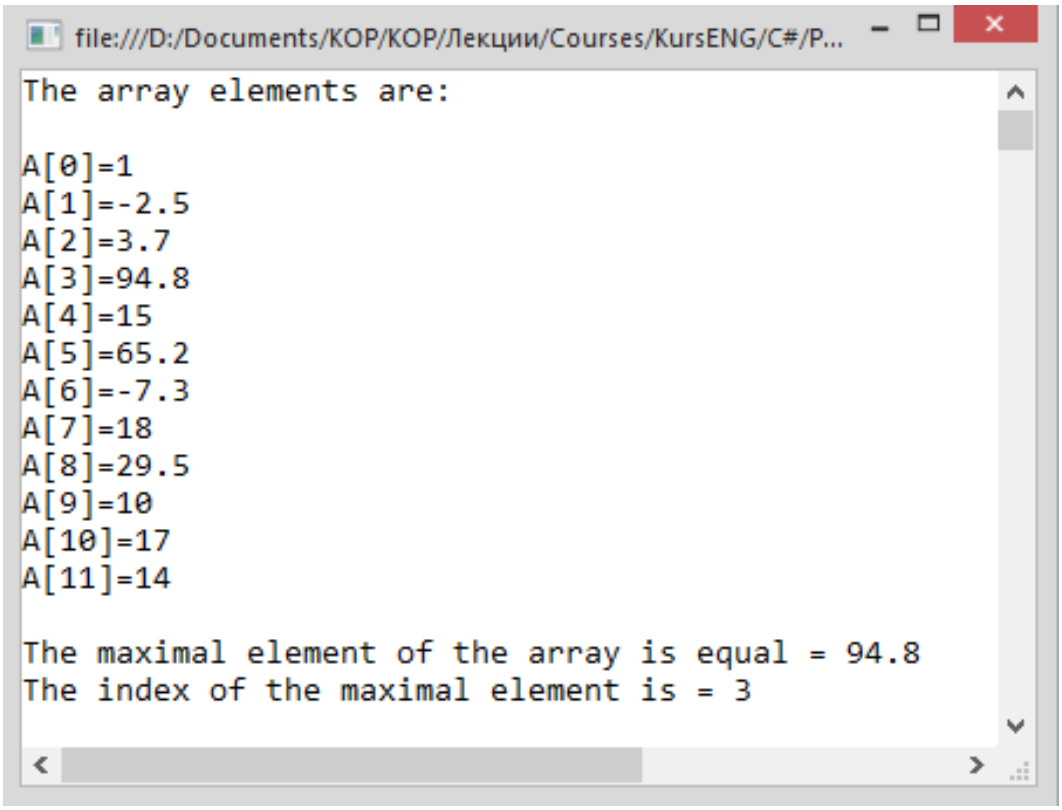


Figure 4.11 - The results of calculations Task 4.5

Task 4.6.

To write the console application to calculate the following expression:

$$P = \sum_{i=0}^{9} \left(x_i + 5,6 y_i \right)$$

where

x_i are the elements of one-dimension array $X(10)$, $i = 0,1,\ldots 9$.

y_i are the elements of vector $Y(10)$, $i = 0,1,\ldots 9$.

The flowchart of the solution for Task 4.6 is presented in Fig. 4.12.

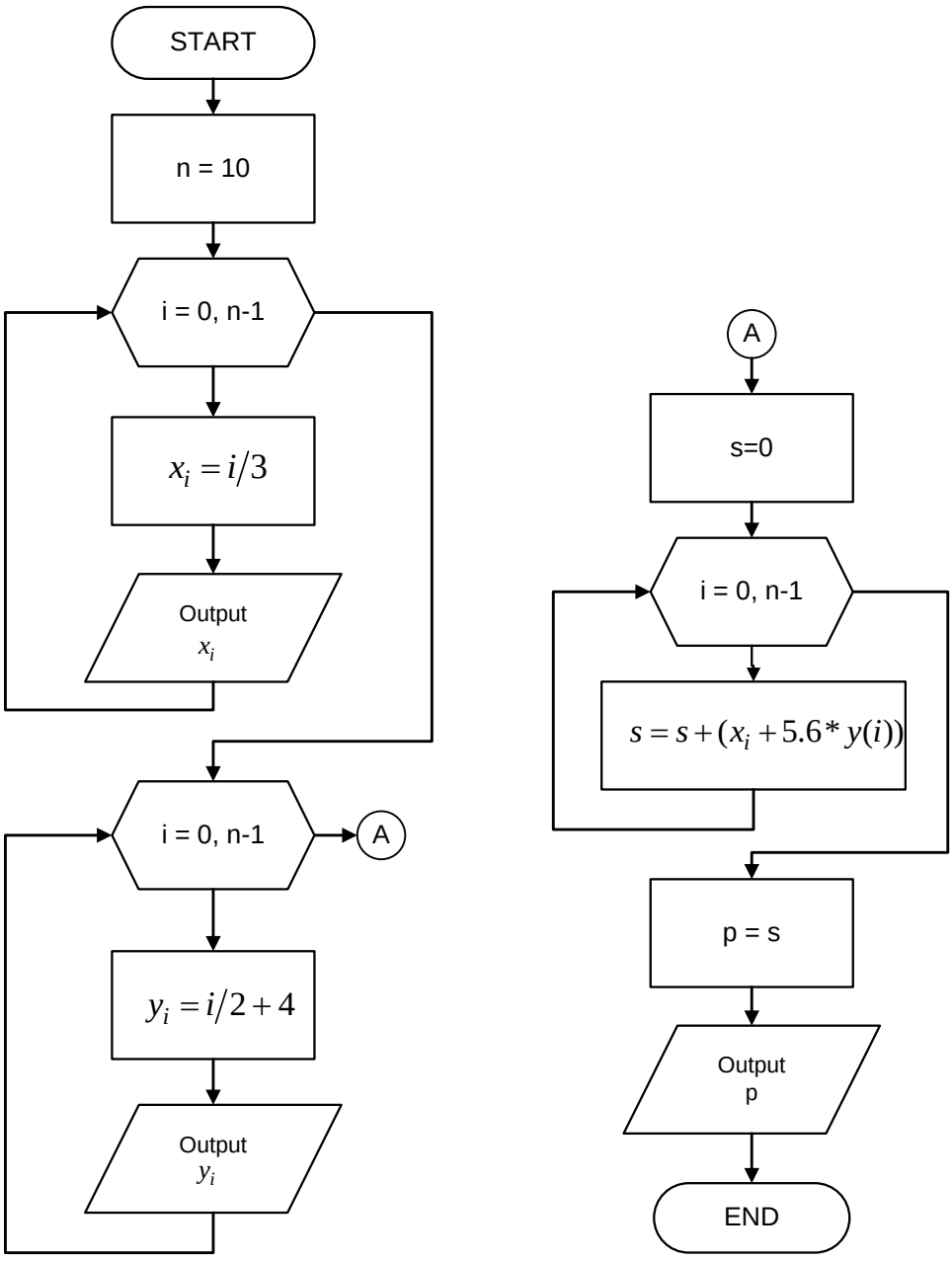


Figure 4.12 - Flowchart of solution for Task 4.6

The following C# source code solves the Task 4.6:

```
using System;
namespace Task4_7
{
    class Program
    {
        static void Main(string[] args)
        {
            const int n = 10;
                  // The number of elements of each array
            double s, p;
            double[] X = new double[n];
            double[] Y = new double[n];
            // The calculation and output of X array
            Console.WriteLine(" Array X ");
            for (int i = 0; i <= (n - 1); i = i + 1)
            {
                X[i] = i / 3.0;
                Console.WriteLine("X[{0}]={1,7:F3} ", i, X[i]);
            }
            Console.WriteLine();
            // The calculation and output of Y array
            Console.WriteLine(" Array Y ");
            for (int i = 0; i <= (n - 1); i = i + 1)
            {
                Y[i] = i / 2.0 + 4;
                Console.WriteLine("Y[{0}]={1,7:F3} ", i, Y[i]);
            }
            Console.WriteLine();
            //Summarizing
            s = 0;
            for (int i = 0; i <= (n - 1); i = i + 1)
            {
                s = s + (X[i] + 5.6 * Y[i]);
            }
            p = s;
            Console.WriteLine("p= {0,7:F3}", p);
```

```
            Console.ReadLine();
        }
    }
}
```

The results of calculations are presented in Fig. 4.13.

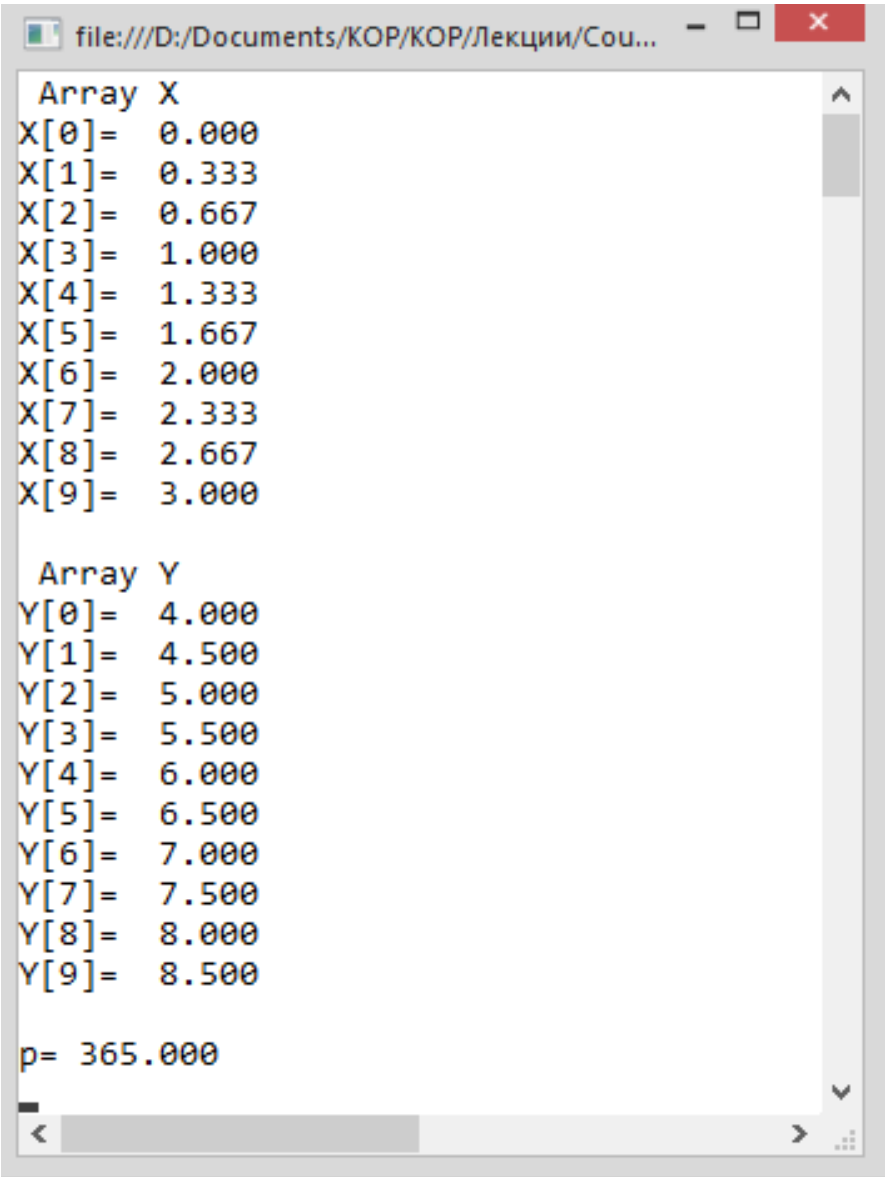


Figure 4.13 - The results of calculations Task 4.6

Task 4.7. Tasks for laboratory works.

1) Create the one-dimension array $A(12)$, where the elements of array a_i are calculated according to formula presented in Table 4.2, column 2.
2) Calculate the sum of the array elements from n to k presented in Table 4.2, column 3.
3) Display the array elements
   – in reverse order (with decreasing indexes) from k to n
   – only elements with even (odd) index of each n-th element of array

Table 4.2 - Variants for calculation

№	Relations for calculation the array elements	Values of n and k
1	2	3
1	$a_i = 1/2^i + 1/3^i;$ $i = 0,1,...11$	$n = 3;\ k = 5$
2	$a_i = (2 \cdot i - 1) \cdot 2^i;$ $i = 0,1,...11$	$n = 2;\ k = 4$
3	$a_i = 1/(3i - 2) \cdot (3i + 1);$ $i = 0,1,...11$	$n = 4;\ k = 6$
4	$a_i = 10^i/(i + 1);$ $i = 0,1,...11$	$n = 5;\ k = 8$
5	$a_i = 10/i^i;$ $i = 0,1,...11$	$n = 2;\ k = 5$
6	$a_i = \ln(i + 2) + i^2;$ $i = 0,1,...11$	$n = 3;\ k = 6$
7	$a_i = i^{\ln(i+1)}/(\ln(5 \cdot i))^i;$ $i = 0,1,...11$	$n = 7;\ k = 9$
8	$a_i = 10/(i + 3);$ $i = 0,1,...11$	$n = 8;\ k = 11$
9	$a_i = e^i/(i + 5);$ $i = 0,1,...11$	$n = 6;\ k = 9$
10	$a_i = (i + 3)^2 \cdot e^i;$ $i = 0,1,...11$	$n = 3;\ k = 6$

Task 4.7 (example)

1) Create the one-dimension array $A(12)$, where the elements of array a_i are calculated according to formula $a_i = 12 + i/\sin(i+1); \quad i = 0,1,...11$
2) Calculate the sum of the array elements from n to k: $n = 5; k = 7$.
3) Display the array elements
   – in reverse order (with decreasing indexes) from k to n
   – only elements with even (odd) index of every n-th element of array

The program is developed as console application.

The following C# source code solves the Task 4.7:

```
using System;

namespace Task4_8
{
    class Program
    {
        static void Main(string[] args)
        {
          const int n = 12; // The number of elements in array
          double[] A = new double[n];
          int i;
          double  s;
          // Calculation and output of array elements
          Console.WriteLine(" Array A: ");
          Console.WriteLine();
          for (i = 0; i <= (n - 1); i = i + 1)
          {
              A[i] = 12 + i / Math.Sin(i + 1);
              Console.WriteLine("A[{0}]={1,7:F4} ", i, A[i]);
          }
          Console.WriteLine();
          // The summation of array elements from 5th to 7th
```

```
            Console.WriteLine("The sum of array elements from 5th
                              to 7th is equal ");
            s = 0;
            for (i = 5; i <= 7; i = i + 1)
                s = s + A[i];
            Console.WriteLine("s= {0,7:F4}", s);
            Console.WriteLine();
            //Reverse output of array elements from 7th to 5th
            Console.WriteLine("Reverse output of array elements
                              from 7th to 5th ");
            for (i = 7; i >= 5; i = i - 1)
                Console.WriteLine("A[{0}]={1,7:F4} ", i, A[i]);
            Console.WriteLine();
            //Display of elements with even index
          Console.WriteLine("The elements with even index are");
            for (i = 2; i <= 10; i = i + 2)
                Console.WriteLine("A[{0}]={1,7:F4} ", i, A[i]);
            Console.ReadLine();
          }
      }
}
```

The results of calculations are presented in Fig. 4.14.

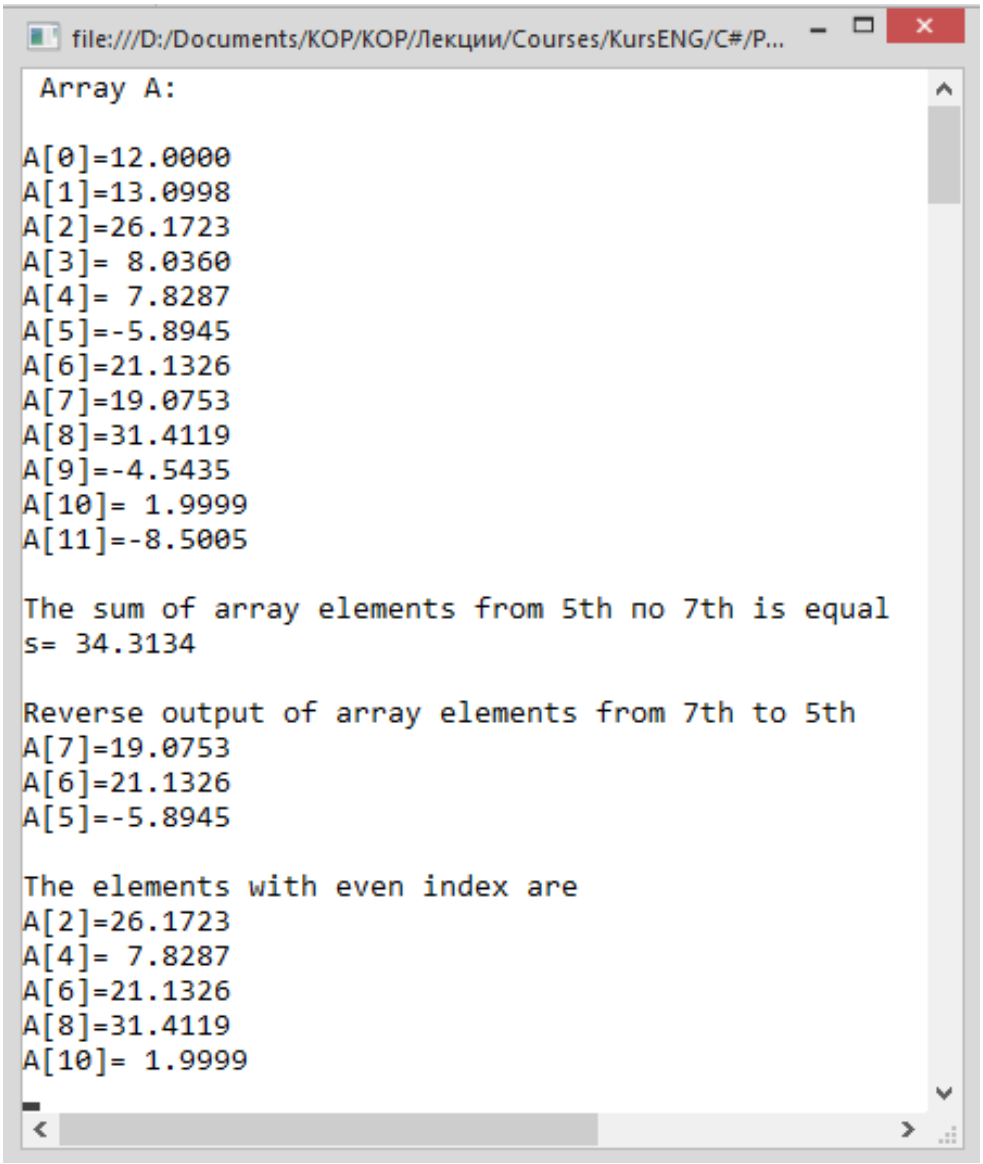


Figure 4.14 - The results of calculations Task 4.7

Questions to Chapter 4.

1. How should a vector be declared in C#?
2. What approaches exist for input and output of vector elements?
3. How should the elements of the vector be filtered?
4. Which methods are applied to find the minimal and maximal element in the vector?
5. Describe the Random Class. When is it used?

Навчальне видання

АРСЕНЬЄВА Ольга Петрівна
СОЛОВЕЙ Людмила Валентинівна

Програмування мовою C#

Навчально-методичний посібник з курсу «Інформатика»
для студентів хімічних спеціальностей

Англійською мовою

Відповідальний за випуск проф. В.Є. Ведь
Роботу до видання рекомендував проф. В.Д. Дмитрієнко

В авторській редакції

План 2019 р., поз. 83.

Підписано до друку 31.05.2019 р. Формат 60×84 1/16. Папір офсетний.
Riso-друк. Гарнітура Таймс. Ум. друк. арк. 5,0 Наклад 20 прим.
Зам № Ціна договірна.

Видавець
Видавничий центр НТУ «ХПІ», вул. Кирпичова, 2, м. Харків-2, 61002
Свідоцтво суб'єкта видавничої справи ДК №5478 від 21.08.2017 р.
