Lab 04 – Operators and Expressions

1. Introduction (Concept Map)

In this lab we will get acquainted with the operators in C# and the actions they can perform when used with the different data types. In the beginning, we will explain which operators have higher priority and we will analyze the different types of operators, according to the number of the arguments they can take and the actions they perform. In the second part, we will examine the conversion of data types. We will explain when and why it is needed to be done and how to work with different data types. At the end of the chapter, we will pay special attention to the expressions and how we should work with them. Finally, we have prepared exercises to strengthen our knowledge of the material in this chapter.

2. Operators

Every programming language uses operators, through which we can perform different actions on the data. Let's take a look at the operators in C# and see what they are for and how they are used.

Operators allow processing of primitive data types and objects. They take as an input one or more operands and return some value as a result. Operators in C# are special characters (such as "+", ".", "^", etc.) and they perform transformations on one, two or three operands. Examples of operators in C# are the signs for adding, subtracting, multiplication and division from math (+, -, *, /) and the operations they perform on the integers and the real numbers.

2.1. Operators in C#

Operators in C# can be separated in several different categories:

- **Arithmetic operators** – they are used to perform simple mathematical operations.
- **Assignment operators** – allow assigning values to variables.
- **Comparison operators** – allow comparison of two literals and/or variables.
- **Logical operators** – operators that work with Boolean data types and Boolean expressions.
- **Binary operators** – used to perform operations on the binary representation of numerical data.
- **Type conversion operators** – allow conversion of data from one type to another.

2.2. Operator Categories

Below is a list of the operators, separated into categories:

Category	Operators
arithmetic	-, +, *, /, %, ++, --
logical	&&, ||, !, ^
binary	&, |, ^, ~, <<, >>
comparison	==,!=, >, <, >=, <=
assignment	=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
string concatenation	+
type conversion	(type), as, is, typeof, sizeof
other	., new, (), [], ?:, ??

2.3. Operator Precedence in C#

Some operators have precedence (priority) over others. For example, in math multiplication has precedence over addition. The operators with a higher precedence are calculated before those with lower. The operator () is used to change the precedence and like in math, it is calculated first. The following table illustrates the precedence of the operators in C#:

Priority	Operators
Highest priority	(,)
	++, -- (as postfix), new, (type), typeof, sizeof
	++, -- (as prefix), +, - (unary), !, ~
	*, /, %
	+ (string concatenation)
	+, -
...	<<, >>
	<, >, <=, >=, is, as
	==, !=
	&, ^, |
Lowest priority	&&
	||
	?:, ??
	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=

The operators located upper in the table have higher precedence than those below them, and respectively they have an advantage in the calculation of an expression. To change the precedence of an operator we

can use brackets. When we write expressions that are more complex or have many operators, it is recommended to use brackets to avoid difficulties in reading and understanding the code. For example:

```
// Ambiguous
x + y / 100

// Unambiguous, recommended
x + (y / 100)
```

2.4. Arithmetical Operators

The arithmetical operators in C# +, -, * are the same like the ones in math. They perform addition, subtraction and multiplication on numerical values and the result is also a numerical value.

The division operator / has different effect on integer and real numbers. When we divide an integer by an integer (like int, long and sbyte) the returned value is an integer (no rounding, the fractional part is cut). Such division is called an integer division. Example of integer division: 7 / 3 = 2.

Integer division by 0 is not allowed and causes a runtime exception DivideByZeroException. The remainder of integer division of integers can be obtained by the ope0rator %. For example, 7 % 3 = 1, and –10 % 2 = 0. When dividing two real numbers or two numbers, one of which is real (e.g. float, double, etc.), a real division is done (not integer), and the result is a real number with a whole and a fractional part. For example: 5.0 / 2 = 2.5. In the division of real numbers it is allowed to divide by 0.0 and respectively the result is +∞ (Infinity), -∞ (-Infinity) or NaN (invalid value).

The operator for increasing by one (increment) ++ adds one unit to the value of the variable, respectively the operator -- (decrement) subtracts one unit from the value. When we use the operators ++ and -- as a prefix (when we place them immediately before the variable), the new value is calculated first and then the result is returned. When we use the same operators as post-fix (meaning when we place them immediately after the variable) the original value of the operand is returned first, then the addition or subtraction is performed. Here are some examples of arithmetic operators and their effect:

00

```
int squarePerimeter = 17;
double squareSide = squarePerimeter / 4.0;
double squareArea = squareSide * squareSide;
Console.WriteLine(squareSide); // 4.25
Console.WriteLine(squareArea); // 18.0625

int a = 5;
int b = 4;
Console.WriteLine(a + b);       // 9
Console.WriteLine(a + (b++));   // 9
Console.WriteLine(a + b);       // 10
Console.WriteLine(a + (++b));   // 11
Console.WriteLine(a + b);       // 11
Console.WriteLine(14 / a);      // 2
Console.WriteLine(14 % a);      // 4

int one = 1;
int zero = 0;
// Console.WriteLine(one / zero); // DivideByZeroException

double dMinusOne = -1.0;
double dZero = 0.0;
Console.WriteLine(dMinusOne / zero); // -Infinity
Console.WriteLine(one / dZero); // Infinity
```

2.5. Logical Operators

Logical (Boolean) operators take Boolean values and return a Boolean result (true or false). The basic Boolean operators are "AND" (&&), "OR" (||), "exclusive OR" (^) and logical negation (!). The following table contains the logical operators in C# and the operations that they perform:

x	y	!x	x && y	x || y	x ^ y
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

The following example illustrates the usage of the logical operators and their actions:

```
bool a = true;
bool b = false;
Console.WriteLine(a && b);              // False
Console.WriteLine(a || b);              // True
Console.WriteLine(!b);                  // True
Console.WriteLine(b || true);           // True
Console.WriteLine((5 > 7) ^ (a == b));  // False
```

2.6. Comparison Operators

Comparison operators in C# are used to compare two or more operands. C# supports the following comparison operators:

- greater than (>)
- less than (<)
- greater than or equal to (>=)
- less than or equal to (<=)
- equality (==)
- difference (!=)

All comparison operators in C# are binary (take two operands) and the returned result is a Boolean value (true or false). Comparison operators have lower priority than arithmetical operators but higher than the assignment operators. The following example demonstrates the usage of comparison operators in C#:

```
int x = 10, y = 5;
Console.WriteLine("x > y : " + (x > y));    // True
Console.WriteLine("x < y : " + (x < y));    // False
Console.WriteLine("x >= y : " + (x >= y));  // True
Console.WriteLine("x <= y : " + (x <= y));  // False
Console.WriteLine("x == y : " + (x == y));  // False
Console.WriteLine("x != y : " + (x != y));  // True
```

2.7. Assignment Operators

The operator for assigning value to a variable is "=" (the character for mathematical equation). The syntax used for assigning value is as it follows:

operand1 = literal, expression or operand2;

Here is an example to show the usage of the assignment operator:

```
int x = 6;
string helloString = "Hello string.";
int y = x;
```

2.7.1. Cascade Assignment

The assignment operator can be used in cascade (more than once in the same expression). In this case assignments are carried out consecutively from right to left. Here's an example:

int x, y, z;

x = y = z = 25;

2.7.2. Compound Assignment

Operators Except the assignment operator there are also compound assignment operators. They help to reduce the volume of the code by typing two operations together with an operator: operation and assignment. Compound operators have the following syntax: Here is an example of a compound operator for assignment:

int x = 2;

int y = 4;

x *= y; // Same as

Console.WriteLine(x); // 8

x = x * y;

Console.WriteLine(x); // 8

2.8. Bitwise Operators

The Bitwise operators supported by C# are listed in the following table. Assume variable A holds 60 and variable B holds 13, then −

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = -61, which is 1100 0011 in 2's complement due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15, which is 0000 1111

Lab tasks

1. Which of the following values can be assigned to variables of type float, double and decimal: 5, -5.01, 34.567839023; 12.345; 8923.1234857; 3456.09112487595654215125668346 7?
2. Create a simple calculator which will perform all arithmetical, Bit wise operation and logical operation on two number
3. Create a simple program to calculate Hypotenuse using Pythagoras theorem

c^2 =(a^2 + b^2)