C# programming language
The beginning

FIIT, Semester 2

Programming Languages

Mikhalkovich S.S.
Abramyan A.V.

Basic operations

```
& | ^ ~              // and, or, xor, not

&& || ^ !            // and, or, xor, not
(i<0 || i==2)        // или
(i>=2 && i!=3)       // и
!(i>2)               // не

min = a < b? a : b;

if (a < b) min = a; else min = b;
```

These are operators with bytes for operands of the integral numeric types.

These are conditional logical operators for logical operands.

The ternary conditional operator

Basic operations

- Increment Operator ++
  - post increment x++;
  - pre increment ++x;

  } x=x+1;
- Decrement Operator --
  - post decrement x--;
  - pre decrement --x;

  } x=x-1;

But, the difference is in the following example. Suppose x=10;

A = x++ - 5; means A=x-5; x=x+1; so, A= 5 and x=11

B =++x - 5; means x=x+1; B=x-5; so, B=6 and x=11

Standard mathematical functions

```
using System;
Math.Abs(x)
Math.Sqrt(x)
Math.Pow(a, b)
Math.Min(a, b)
Math.Max(a, b)
Math.Sin(x)
Math.Cos(x)
Math.Exp(x)
Math.Log(x)
```

```
using System;
Math.Round(2.6)=3.0
Math.Round(2.5)=2.0
Math.Round(3.5)=4.0
Math.Floor(2.6)=2.0
Math.Ceiling(2.4)=3.0
```

Random number generation

```
Random r = new Random();
var i = r.Next(2, 5);
var r = r.NextDouble();
```

Implicit Casting and Explicit Casting

Implicit Casting done automatically when passing a smaller size type to a larger size type.

```
char -> int -> long -> float -> double

int myInt = 9;
double myDouble = myInt;       // Automatic casting: int to double
```

Explicit Casting must be done manually by placing the type in parentheses in front of the value.

```
double -> float -> long -> int -> char

double myDouble = 9.78;
int myInt = (int) myDouble;    // Manual casting: double to int
```

Type Conversion Methods

It is also possible to convert data types explicitly by using built-in methods:

- Convert.ToBoolean,
- Convert.ToDouble,
- Convert.ToString,
- Convert.ToInt32 (int),
- Convert.ToInt64 (long).

```
int myInt = 10;
double myDouble = 5.25;
bool myBool = true;

var a = Convert.ToString(myInt);    // convert int to string a = "10"
var b = Convert.ToDouble(myInt);    // convert int to double  b = 10.0
var c = Convert.ToInt32(myDouble);  // convert double to int  c = 5
var d = Convert.ToString(myBool);   // convert bool to string d = "True"
```

User Input

The Console.ReadLine() method returns a string.

So you must convert any type explicitly,
by using one of the Convert.To methods.

```
Console.WriteLine("Enter your age:");
int age = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Your age is: " + age);
```

Conditional and switch statements

```
if (a<b)
  min = a;
else min = b;
```

```
if (i==0 && i!=1)
  i++;
else i--;
```

```
if (x<y)
{
  var t = x;
  x = y;
  y = t;
}
```

```
switch (i)
{
  case 1:
    WriteLine("First");
    break;
  case 2:
  case 3:
    WriteLine("Second");
    break;
  default:
    WriteLine("Third");
    break;
}
```

Loops

```
i=5; j=0;
while (i>0)
{
  i--;
  j++;
}
```

```
i=5; j=0;
while (i>0)
{
  i--;
  j++;
}
```

```
for (int i=0; i<10; i++)
  Write($"{i} ");
```

```
for (var i=1; i<100; i*=2)
  Write($"{i} ");
```

```
foreach (var x in a)
  Write($"{x} ");
```

Methods

```
class Program
{
    static void Main()
    {
        Console.WriteLine(Min(3,2));
        Console.WriteLine(MyFuncs.Add(3, 2));
        MyFuncs.PrintInfo();
    }
    static int Min(int a, int b)
    {
        if (a < b)
            return a;
        else return b;
    }
}

class MyFuncs
{
    public static int Add(int i, int j) => i + j;
    public static void PrintInfo() => Console.WriteLine("Info");
}
```

Methods

- Methods can be defined as class methods only
- If method doesn't return a value then type "void" is used
- We use a "return" statement to return a value from a method
- We can define method in the same class as Main() function or in another class
- We can use short definition of a method using =>
- All methods must be static. It means that you can call them without a creation of an object
- Methods defined in one class are visible in another one only if they are defined with public modifier
- If method is called in the same class, the "public" modifier is not required

Parameters, passed by reference

- Parameters, passed by reference, must be **out** (output) or **ref** (input-output)
- A special feature of **ref** is that the variable that we pass to the method must be initialized with a value, otherwise the compiler will throw the error
  `"Use of unassigned local variable 'a' "`
- The use keyword **out** is used in exactly the same way as **ref**, except that the parameter does not have to be initialized before passing, but the passed parameter must be assigned a new value in the method.
- Keywords **out** and **ref** must be using in call:

```
Square(5, out res);
Swap(ref a, ref b);
```

- We can define out-parameter in a call place (!):

```
Square(5, out var res1);
```

Generic Methods

```
class Program
{
    static void Main()
    {
        int res;
        Square(5, out res);
        Square(5, out var res1); //

        (int a, int b) = (3, 5); //
        Swap(ref a, ref b);
    }
    static void Swap<T>(ref T a, ref T b)
    {
        var x = a;
        a = b;
        b = x;
    }
    static void Square(int a, out int res)
    {
        res = a * a;
    }
}
```

In C# the Swap method is absent (!)

Enum type

```
enum Color { Red, Green, Blue };

string Name(Color c)
{
    switch (c)
    {
        case Color.Red: return "Red";
        case Color.Green: return "Green";
        case Color.Blue: return "Blue";
        default: return "No color";
    }
}
```

Sequences

```
// C#
IEnumerable<int> sq;
sq = System.Linq.Enumerable.Range(1,10);
foreach (var x in sq)
  Console.Write(Console.Write($"{x} ");
```

```
// PascalABC.NET
var sq: sequence of integer;
sq = Range(1,10);
sq.Print;
```