

Jazyk C# a .NET

prednáška

3



ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

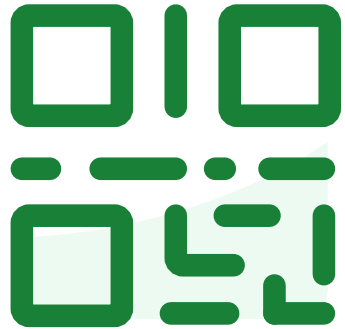
Základy jazyka C# (2)

Ing. **Štefan Toth**, PhD.

06.03.2025

slido

Please download and install the Slido app on all computers you use



**Join at slido.com
#2518723**

① Start presenting to display the joining instructions on this slide.

- **Polia** (arrays)
- **Indexy a rozsahy** (indices and ranges)
- **Reťazce** (strings)
- **Menné priestory** (namespaces)
- **Argumenty príkazového riadka** (command-line arguments)
- **NuGet balíčky** (NuGet packages)

Polia (1)

- **Jednorozmerné** (single-dimensional):
 - `type[]` jednorozmerne
- **Viacrozmerné** (multidimensional):
 - `type[,]` dvojrozmerne
 - `type[, ,]` trojrozmerne
 - `type[, , ,]` stvorrozmerne
 - ... až max 32 rozmerné
- **Pole polí / „zubaté“ polia** (jagged):
 - `type[][]` ortogonale
 - `type[][,]` ortogonale2 // Mix „zubaté“ s 2-rozmerným
 - `type[][][]` ortogonale3
 - ...
- Základom všetkých polí je abstraktná trieda **System.Array**

Polia (2) – jednorozmerné polia

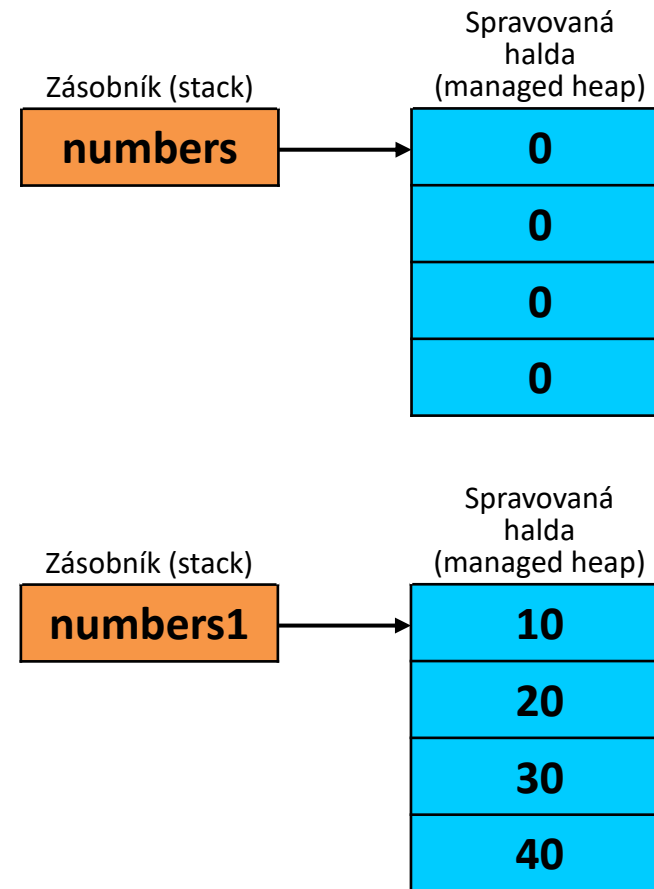
- Pole je referenčný typ, pamäť sa alokuje na halde (heap), premenná v metóde na zásobníku (stack)

```
// Deklarácia poľa  
int[] numbers;
```

```
// Deklarácia a alokovanie poľa  
int[] numbers = new int[4];
```

```
// Inicializácia polí  
int[] numbers1 = new int[4] { 10, 20, 30, 40 };  
int[] numbers2 = new int[] { 10, 20, 30, 40 };  
int[] numbers3 = { 10, 20, 30, 40 };  
int[] numbers4 = [10, 20, 30, 40]; // Od C# 12.0 / .NET 8
```

- Potom, čo sa špecifikuje veľkosť poľa, už veľkosť nemôže byť zmenená
 - musí sa alokovať nové pole a skopírovať jeho prvky do nového poľa



Polia (3) – jednorozmerné polia – prístup a zápis

- Príklad jednorozmerného poľa:

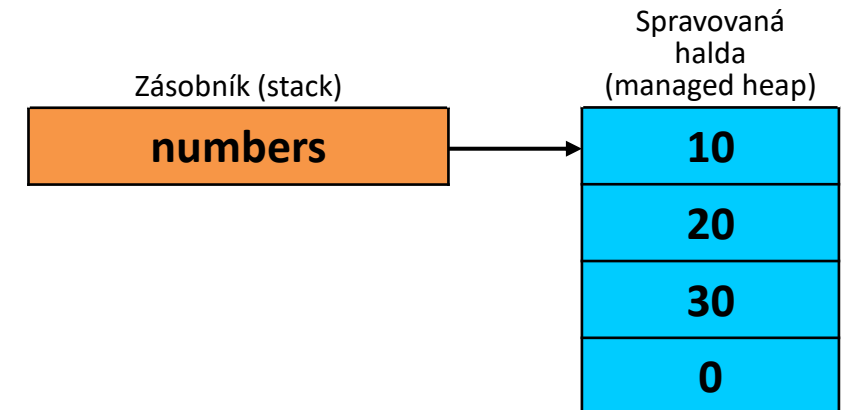
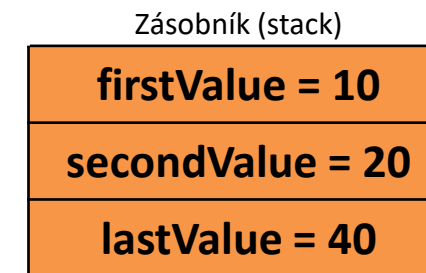
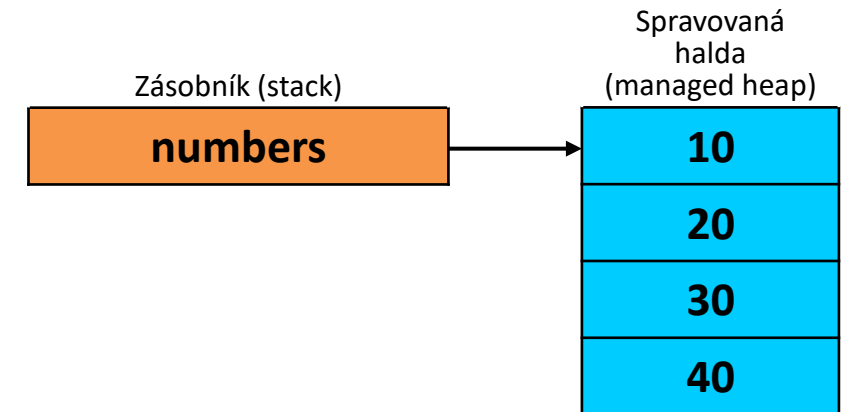
```
var numbers = new int[] { 10, 20, 30, 40 };
```

- Na **prístup** / prečítanie hodnoty sa používa **indexer** s hranatými zátvorkami **[]**:

```
int firstValue = numbers[0]; // Vráti hodnotu 10  
int secondValue = numbers[1];  
int lastValue = numbers[numbers.Length - 1];  
int value = numbers[4]; // IndexOutOfRangeException
```

- Na **zápis** / prepis hodnôt sa používa tiež indexer:

```
numbers[3] = 0; // Na indexe 3 prepíše hodnotu na 0
```



Polia (4) – jednorozmerné polia – iterovanie

- Iterovanie cez **for** (vlastnosť **Length** udáva veľkosť poľa typu int; ak by ste mali väčšie pole, môžete použiť aj vlastnosť **LongLength**):

```
for (int i = 0; i < numbers.Length; i++)  
{  
    Console.WriteLine(numbers[i]);  
}
```

```
for (long i = 0; i < numbers.LongLength; i++)  
{  
    Console.WriteLine(numbers[i]);  
}
```

- Iterovanie cez **foreach**:

```
foreach (var value in numbers)  
{  
    Console.WriteLine(value);  
}
```

Polia (5) – použitie referenčných typov

- Ak použijeme referenčný typ, napr. record class:

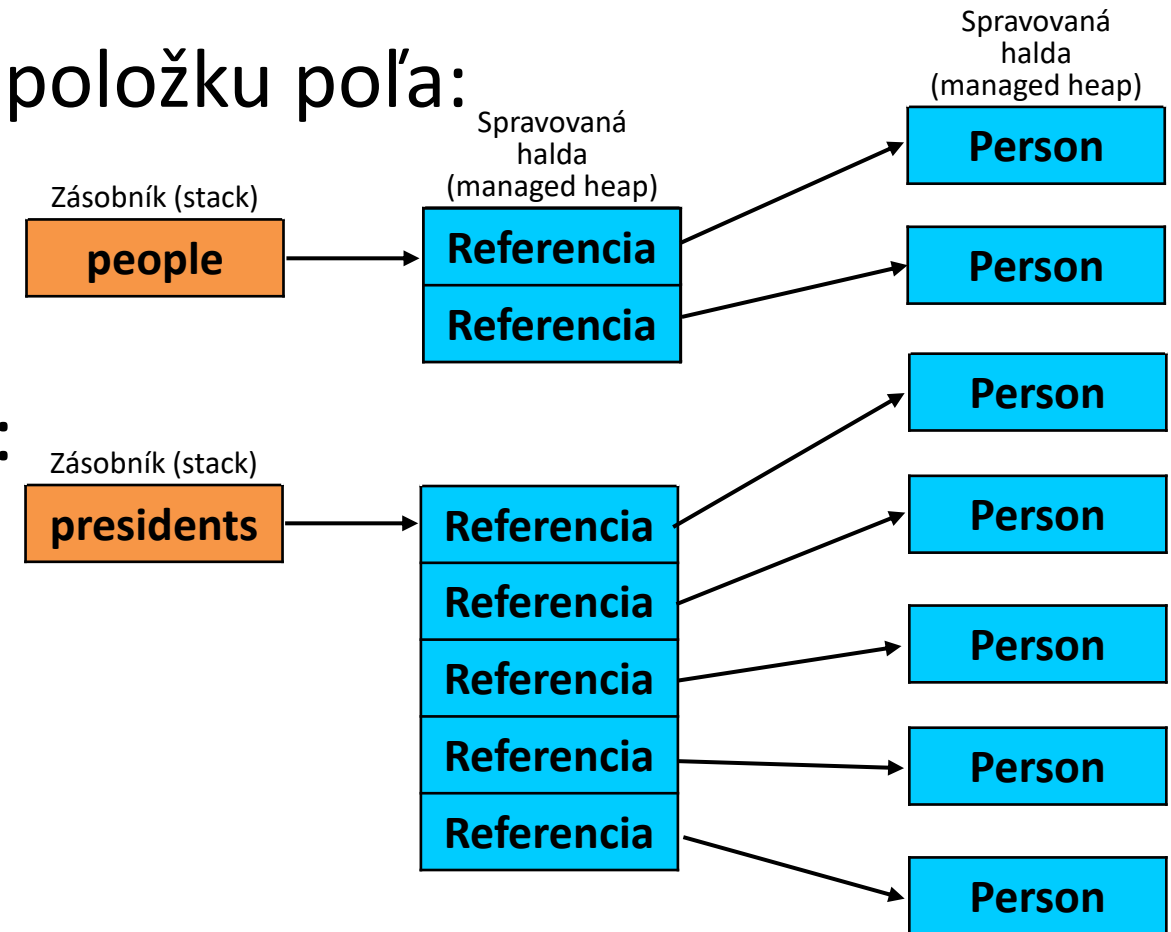
```
public record class Person(string FirstName, string LastName);
```

- Musíme vytvoriť objekt pre každú položku poľa:

```
Person[] people;  
people = new Person[2];  
people[0] = new Person("Jan", "Mrkvicka");  
people[1] = new("Fero", "Mrkvicka");
```

- Alebo použijeme inicializátor poľa:

```
Person[] presidents =  
{  
    new("Michal", "Kováč"),  
    new("Rudolf", "Schuster"),  
    new("Ivan", "Gašparovič"),  
    new("Andrej", "Kiska"),  
    new("Zuzana", "Čaputová"),  
}
```



Polia (6) – viacrozmerné polia

```
// Vytvorenie a naplnenie  
2-rozmerného poľa o veľkosti 3x3:  
int[,] twodim = new int[3, 3];  
twodim[0, 0] = 1;  
twodim[0, 1] = 2;  
twodim[0, 2] = 3;  
twodim[1, 0] = 4;  
twodim[1, 1] = 5;  
twodim[1, 2] = 6;  
twodim[2, 0] = 7;  
twodim[2, 1] = 8;  
twodim[2, 2] = 9;
```

	0.	1.	2.
0.	1	2	3
1.	4	5	6
2.	7	8	9

```
// Vytvorenie a inicializácia  
// 2-rozmerného poľa  
int[,] twodim =  
{  
    { 1, 2, 3 },  
    { 4, 5, 6 },  
    { 7, 8, 9 }  
};
```

```
// Príklad pre 3-rozmerné pole  
int[, ,] threedim =  
{  
    { { 1, 2 }, { 3, 4 } },  
    { { 5, 6 }, { 7, 8 } },  
    { { 9, 10 }, { 11, 12 } }  
};
```

			2.	0.	1.
			0.	9	10
		1.	0.	5	6
		0.	1.	7	8
0.	0.	1.			
0.	1	2			
1.	3	4			

Polia (7) – viacrozmerné polia – výpis

```
int[,] twodim =  
{  
    { 1, 2, 3 },  
    { 4, 5, 6 },  
};
```

	0.	1.	2.
0.	1	2	3
1.	4	5	6

```
foreach (var value in twodim)  
{  
    Console.WriteLine(value);  
}
```

1
2
3
4
5
6

```
Console.WriteLine($"Rank: {twodim.Rank}"); // Počet rozmerov  
Console.WriteLine($"Length: {twodim.Length}"); // Počet prvkov  
Console.WriteLine($"GetLength(0): {twodim.GetLength(0)}");  
Console.WriteLine($"GetLength(1): {twodim.GetLength(1)}");
```

```
for (int col = 0; col < twodim.GetLength(0); col++)  
{  
    for (int row = 0; row < twodim.GetLength(1); row++)  
    {  
        Console.Write(twodim[col, row] + " ");  
    }  
  
    Console.WriteLine();  
}
```

```
Rank: 2  
Length: 6  
GetLength(0): 2  
GetLength(1): 3  
1 2 3  
4 5 6
```

Polia (8) – pole polí

```
int[][] jagged = new int[3][];  
jagged[0] = new int[] { 1, 2 };  
jagged[1] = new int[] { 3, 4, 5, 6, 7, 8 };  
jagged[2] = new int[] { 9, 10, 11 };
```

```
Console.WriteLine($"Rank: {jagged.Rank}");  
Console.WriteLine($"Length: {jagged.Length}");  
Console.WriteLine($"GetLength(0): {jagged.GetLength(0)}");
```

```
for (int i = 0; i < jagged.Length; i++)  
{  
    System.Console.Write($"Element({i}): ");  
  
    for (int j = 0; j < jagged[i].Length; j++)  
    {  
        System.Console.Write(jagged[i][j] + " ");  
    }  
    System.Console.WriteLine();  
}
```

```
int[][] jagged =  
{  
    new[] { 1, 2 },  
    new[] { 3, 4, 5, 6, 7, 8 },  
    new[] { 9, 10, 11 }  
};
```

	0.	1.	2.	3.	4.	5.
0.	1	2				
1.	3	4	5	6	7	8
2.	9	10	11			

```
Rank: 1  
Length: 3  
GetLength(0): 3  
Element(0): 1 2  
Element(1): 3 4 5 6 7 8  
Element(2): 9 10 11
```

slido

Please download and install the Slido app on all computers you use



**Ako sa dostaneme k poslednému
prvku poľa myArray v jazyku C#?**

① Start presenting to display the poll results on this slide.

slido

Please download and install the Slido app on all computers you use



Aké hodnoty vlastností Rank a Length bude mať nasledujúce pole: `int[,] array = { { { 1, 2 }, { 4, 5 } }, { { 6, 7 }, { 8, 9 } }, { { 10, 11 }, { 12, 13 } } };`

① Start presenting to display the poll results on this slide.

Trieda Array (1)

- **Základná trieda pre všetky polia**, ktoré sa vytvoria

```
public abstract class Array : ICloneable, IList, IStructuralComparable, IStructuralEquatable
```

- Poskytuje metódy pre **vytváranie, manipuláciu, vyhľadávanie a triedenie** polí
- Keďže je abstraktná, nie je možné vytvárať priamo jej inštancie, jedine cez jej statickú metódu: **Array.CreateInstance()**
- Vytvorenie plytkej kópie (shallow copy): **Array.Clone()**
- Triedenie: **Array.Sort()** – využíva Quicksort algoritmus na utriedenie jednorozmerného poľa
 - vyžaduje od prvkov implementáciu rozhrania IComparable (jednoduché typy ako System.String alebo System.Int32 toto rozhranie už implementujú) alebo objektu porovnávača, ktorý implementuje rozhranie IComparer

Trieda Array (2) – najpoužívannejšie metódy

- **BinarySearch()** – vyhľadáva prvok v utriedenom jednorozmernom poli
- **IndexOf()** – vracia index prvého výskytu zadanej hodnoty
- **LastIndexOf()** – vracia index posledného výskytu zadanej hodnoty
- **Exists()** – skontroluje, či zadaná hodnota existuje v poli
- **CopyTo()** – kopíruje obsah poľa do iného poľa
- **Reverse()** – obráti poradie prvkov v poli
- **Clear()** – nastaví všetky prvky poľa na predvolenú hodnotu
- **Resize()** – zmení veľkosť poľa
- **FindAll()** – vráti pole obsahujúce všetky prvky, ktoré spĺňajú zadanú podmienku
- **ForEach()** – prejde cez všetky prvky poľa a vykoná zadanú akciu pre každý prvok
- **GetValue()** – vráti hodnotu prvku poľa
- **SetValue()** – nastaví hodnotu prvku poľa
- ...

Polia – Array.CreateInstance (1)

- Vytvárať nové inštanície polí je možné aj cez statickú metódu **CreateInstance** triedy **Array**
 - Je to užitočné, ak **nepoznáme dopredu typ prvkov** alebo **chceme vytvoriť polia, ktoré začínajú od určitého indexu** iného než 0

```
// Vytvorí pole typu int s počtom prvkov 5
var intArray = Array.CreateInstance(typeof(int), 5);
for (int i = 0; i < intArray.Length; i++)
{
    intArray.SetValue((i + 1) * 10, i); // Zapiše hodnotu (i+1)*10 na index i
    Console.Write(intArray.GetValue(i) + " "); // Prečíta hodnotu na indexe i
}
```

10 20 30 40 50

```
foreach (var number in intArray)
    Console.Write(number + " ");
```

10 20 30 40 50

```
int[] numbers = (int[])intArray;
Console.WriteLine(numbers[0]);
```

10

Kompilačná chyba CS0021:
Cannot apply indexing with [] to
an expression of type 'Array'

```
Console.WriteLine(intArray[0]);
```


Polia – Array.CreateInstance (2)

- Príklad poľa, ktoré začína od iného indexu:

```
int[] lengths = { 2, 3 }; // Prvý rozmer poľa: 2, Druhý rozmer: 3
int[] lowerBounds = { 1, 10 }; // Začiatkové indexy polí
Array array = Array.CreateInstance(typeof(string), lengths, lowerBounds);
array.SetValue("A", 1, 10);
```

```
string[,] matrix = (string[,])array;
matrix[1, 11] = "B";
matrix[1, 12] = "C";
```

```
matrix[2, 10] = "D";
matrix[2, 11] = "E";
matrix[2, 12] = "F";
```

	10.	11.	12.
1.	A	B	C
2.	D	E	F

matrix	{string[1..2, 10..12]}
[1, 10]	View "A"
[1, 11]	View "B"
[1, 12]	View "C"
[2, 10]	View "D"
[2, 11]	View "E"
[2, 12]	View "F"

- Pozor ale potom na hranice poľa:

```
matrix[0, 0] = "X";
```

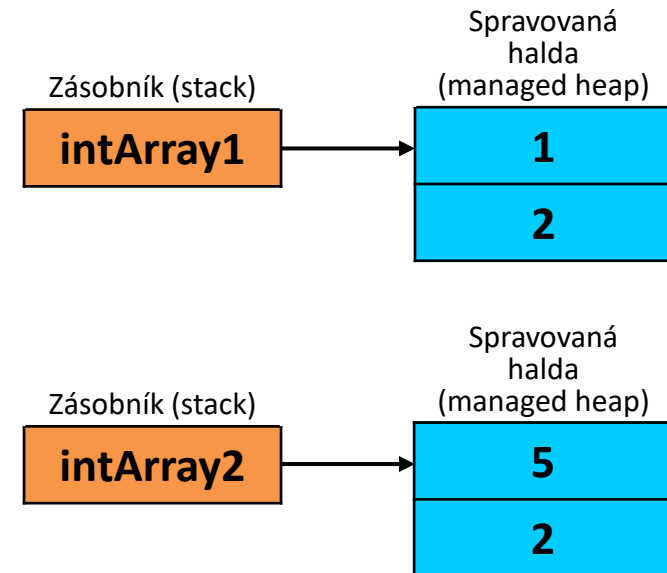
Výnimka za behu: **System.IndexOutOfRangeException**: 'Index was outside the bounds of the array.'

Kopírovanie polí (1) – hodnotové typy

- Polia implementujú rozhranie `ICloneable`, ktoré obsahuje metódu **`Clone()`** vytvárajúcu **plytkú (shallow) kópiu** poľa

```
int[] intArray1 = { 1, 2 };  
int[] intArray2;  
intArray2 = (int[])intArray1.Clone();  
  
intArray2[0] = 5;  
Console.WriteLine(intArray1[0]);
```

1



Kopírovanie polí (2) – referenčné typy

```
class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }

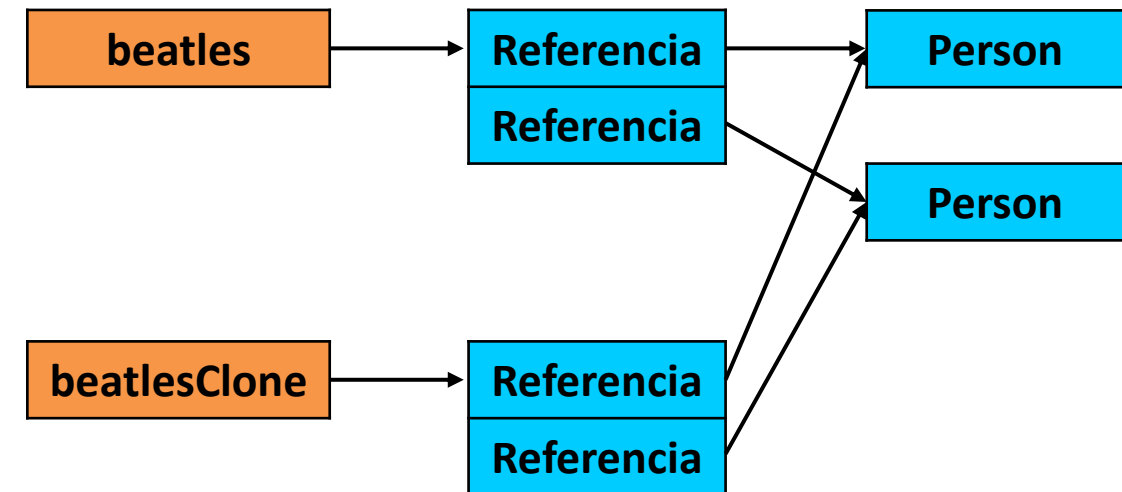
    public Person(string firstName, string lastName)
    {
        FirstName = firstName;
        LastName = lastName;
    }
}

Person[] beatles =
{
    new("John", "Lenon"),
    new("Paul", "McCartney")
};

Person[] beatlesClone = (Person[])beatles.Clone();

beatlesClone[0].LastName = "Mrkvička";
Console.WriteLine(beatles[0].LastName);
```

Mrkvička



Indexy a rozsahy (indices and ranges) (1)

- **Rozsahy** (ranges) a **indexy** (indices) umožňujú jednoducho prístup k prvkom alebo oblastiam v postupnosti
 - Operátor klobúka (hat) **^**
 - Operátor rozsahu (range) **..**
 - Štruktúra **Index**
 - Prvý prvok: **[0]**
 - Posledný prvok: **[^1]**
 - Štruktúra **Range** – rozsah:
 - **Od začiatku – uzavretý** interval
 - **Do konca – otvorený** interval
 - Od prvého po posledný prvok: **[..]** alebo **[0..^0]** – pozor: koniec je preto **^0**, lebo je definovaný s otvoreným intervalom! **^0** je prvok po poslednom prvku

```
int[] data = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };  
  
int first1 = data[0];  
int last1 = data[data.Length - 1]; // Tradičný starý spôsob  
Console.WriteLine($"Prvy: {first1}, posledny: {last1}");  
int last2 = data[^1]; // Spôsob cez operátor klobúka ^  
Console.WriteLine($"Posledny znova: {last2}");
```

```
Index firstIndex = 0;  
Index lastIndex = ^1;  
int first3 = data[firstIndex];  
int last3 = data[lastIndex];  
Console.WriteLine($"Prvy: {first3}, posledny: {last3}");
```

Prvy: 1, posledny: 9
Posledny znova: 9
Prvy: 1, posledny: 9

Indexy a rozsahy (indices and ranges) (2)

```
int[] data = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

```
ShowRange("Cely rozsah", data[..]); // Alebo to isté: [0..] alebo [..^0] alebo [0..^0]
```

```
ShowRange("Prve tri", data[0..3]); // Alebo to isté: [..3]
```

```
ShowRange("Od indexov v intervale <3, 6)", data[3..6]);
```

```
ShowRange("Od indexov v intervale <Length - 3, Length - 0)", data[^3..^0]); // Alebo [^3..]
```

```
// Lokálna funkcia na vypísanie prvkov
```

```
void ShowRange(string title, int[] data)
```

```
{
```

```
    Console.WriteLine($"{title}:");
```

```
    Console.WriteLine(string.Join(" ", data));
```

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

```
}
```

Cely rozsah:

1 2 3 4 5 6 7 8 9

Prve tri:

1 2 3

Od indexov v intervale <3, 6):

4 5 6

Od indexov v intervale <Length - 3, Length - 0):

7 8 9

Indexy a rozsahy (indices and ranges) (3)

- Pozor: použitím rozsahu sa prvky poľa kopírujú, preto zmena hodnoty v rozsahu sa neprejaví v pôvodnom poli:

```
int[] data = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

```
Range fullRange = ..;
```

```
Range firstThree = 0..3;
```

```
Console.WriteLine(fullRange);
```

```
ShowRange("Cely rozsah", data[fullRange]);
```

```
ShowRange("Prve tri", data[firstThree]);
```

```
var slice = data[3..5];
```

```
slice[0] = 777;
```

```
Console.WriteLine($"Hodnota v poli sa nezmenila: {data[3]}, hodnota z rezu: {slice[0]}");
```

0..^0

Cely rozsah:

1 2 3 4 5 6 7 8 9

Prve tri:

1 2 3

Hodnota v poli sa nezmenila: 4, hodnota z rezu: 777

Span<T> a ReadOnlySpan<T> (1)

- **Span<T>** – ref štruktúra, ktorá sa používa pre rýchly a priamy **prístup k súvislej oblasti ľubovoľnej časti** manažovanej a nemanážovanej **pamäte**
- **ReadOnlySpan<T>** – ref štruktúra, rovnako ako Span<T>, ale k dispozícii iba na čítanie z pamäte

```
int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };  
Span<int> span = new(arr);  
span[1] = 777;  
Console.WriteLine($"Hodnota v poli sa zmenila: {arr[1]}");  
DisplaySpan("span", span);  
DisplaySpan("arr", arr);
```

```
void DisplaySpan(string title, ReadOnlySpan<int> span)  
{  
    Console.WriteLine($"{title}:");  
    for (int i = 0; i < span.Length; i++)  
    {  
        Console.Write($"{span[i]} ");  
    }  
    Console.WriteLine();  
}
```

```
Hodnota v poli sa zmenila: 777  
span:  
1 777 3 4 5 6 7 8 9  
arr:  
1 777 3 4 5 6 7 8 9
```

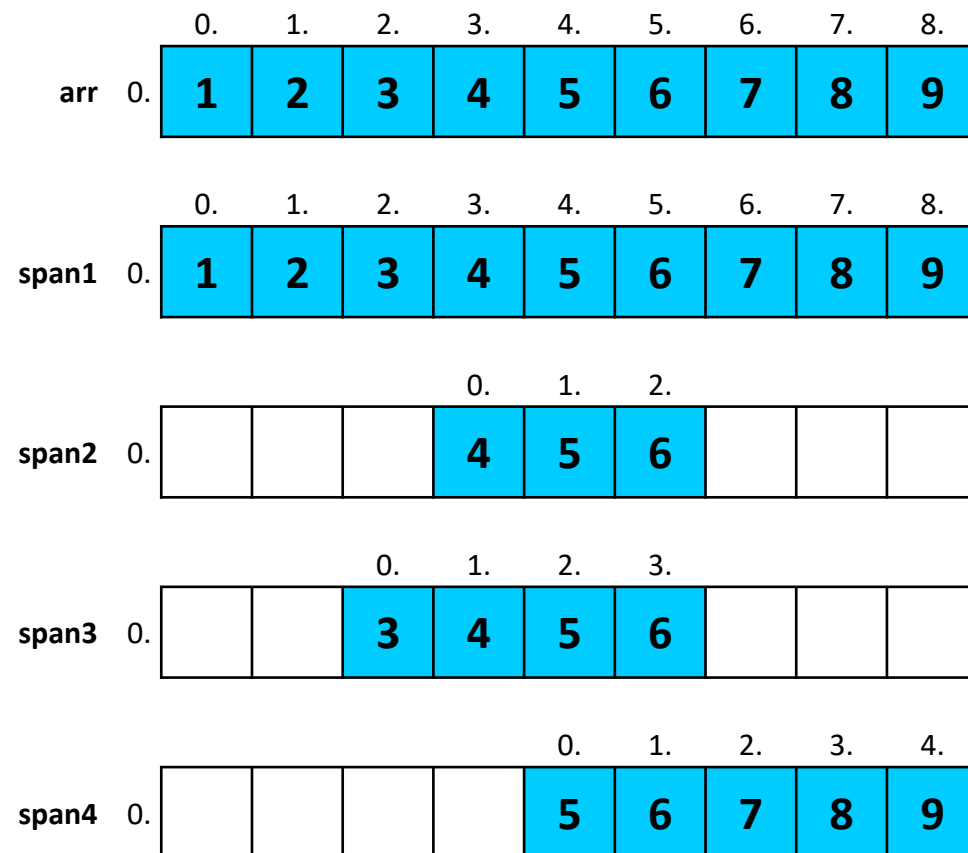
Span<T> a ReadOnlySpan<T> (2) – rezy (slices)

- K častiam poľa sa môžeme dostať pomocou tzv. **rezov (slices)** – vtedy nie sú prvky poľa kopírované, ale prístupuje sa k nim priamo

```
int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };  
Span<int> span1 = new(arr);  
Span<int> span2 = new(arr, start: 3, length: 3);  
Span<int> span3 = span1.Slice(start: 2, length: 4);  
Span<int> span4 = span1.Slice(start: 4);
```

```
DisplaySpan("span1", span1);  
DisplaySpan("span2", span2);  
DisplaySpan("span3", span3);  
DisplaySpan("span4", span4);
```

```
span1:  
1 2 3 4 5 6 7 8 9  
span2:  
4 5 6  
span3:  
3 4 5 6  
span4:  
5 6 7 8 9
```



Span<T> – zmena hodnôt

- **Clear()** – vyplní hodnoty 0
- **Fill()** – vyplní hodnoty zvolenou hodnotou
- **CopyTo()** – skopíruje hodnoty do zvoleného „spanu“
- **TryCopyTo()** – vráti true, ak je možné skopírovať hodnoty vzhľadom na veľkosť cieľového „spanu“ (cieľový „span“ musí byť väčší alebo rovný ako zdrojový)

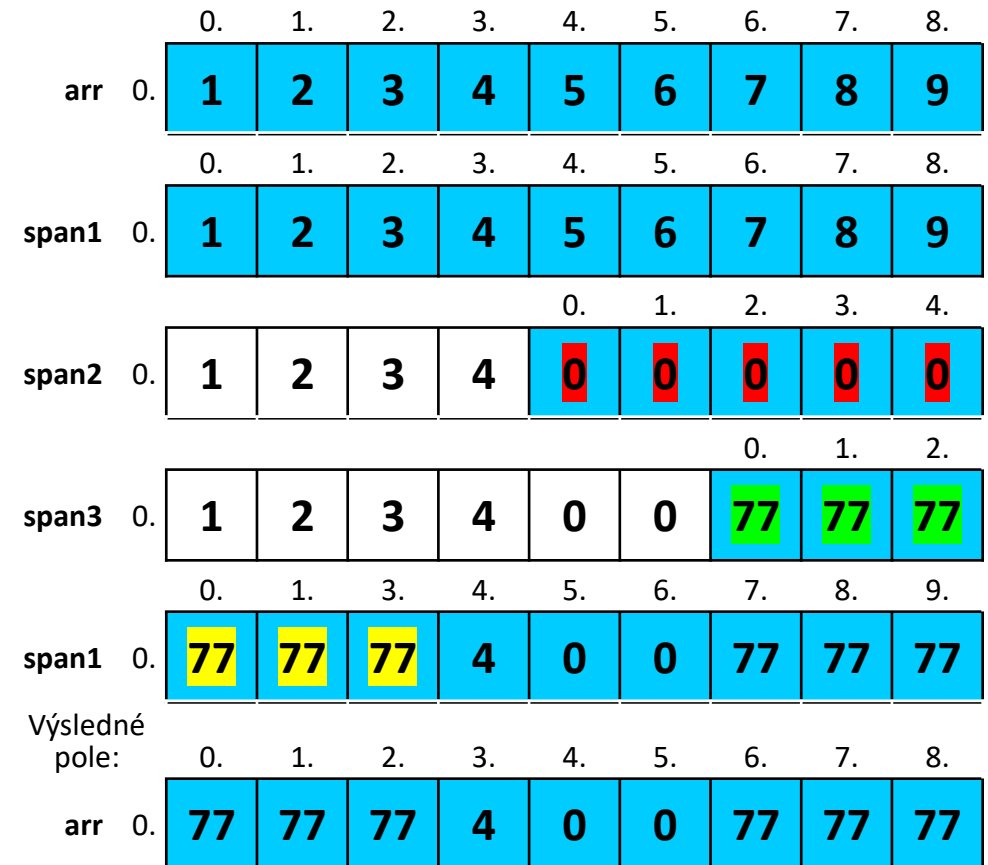
```
int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };  
Span<int> span1 = new(arr);
```

```
Span<int> span2 = span1.Slice(start: 4);  
span2.Clear();  
DisplaySpan("span1 po Clear", span1);
```

```
Span<int> span3 = span2.Slice(start: 2, length: 3);  
span3.Fill(77);  
DisplaySpan("span1 po Fill", span1);
```

```
span3.CopyTo(span1);  
DisplaySpan("span1 po CopyTo", span1);
```

```
span1 po Clear:  
1 2 3 4 0 0 0 0 0  
span1 po Fill:  
1 2 3 4 0 0 77 77 77  
span1 po CopyTo:  
77 77 77 4 0 0 77 77 77
```



Span<T> – použitie s Range

```
int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

```
Span<int> span1 = new(arr);
```

```
Span<int> span2 = span1[4..];
```

```
span2[^1] = 0;
```

```
DisplaySpan("span1", span1);
```

```
DisplaySpan("span2", span2);
```

```
Span<int> span3 = arr.AsSpan()[2..6];
```

```
span3[0] = 0;
```

```
DisplaySpan("span3", span3);
```

```
span1:  
1 2 3 4 5 6 7 8 0  
span2:  
5 6 7 8 0  
span3:  
0 4 5 6
```

	0.	1.	2.	3.	4.	5.	6.	7.	8.	
arr	0.	1	2	3	4	5	6	7	8	9

		0.	1.	2.	3.	4.	5.	6.	7.	8.
span1	0.	1	2	3	4	5	6	7	8	9

					0.	1.	2.	3.	4.	
span2	0.	1	2	3	4	0	6	7	8	0

				0.	1.	2.	3.			
span3	0.	1	2	0	4	5	6	7	8	0

Výsledné pole:	0.	1.	2.	3.	4.	5.	6.	7.	8.
arr 0.	1	2	0	4	5	6	7	8	0

stackalloc

- **Alokuje blok pamäte v zásobníku** pre efektívny a veľmi rýchly prístup k prvkom (rýchlejší než na halde), po skončení metódy je pamäť automaticky uvoľnená, nespravuje ho GC (garbage collector)
- Obmedzenia:
 - dostupné len v rámci metódy, v ktorej bola pamäť alokovaná
 - **množstvo pamäte** na zásobníku je **obmedzené** (záleží od prostredia) – pozor na výnimku pretečenia zásobníka `StackOverflowException`
 - výsledok môže byť priradený iba do: **`Span<T>`**, **`ReadOnlySpan<T>`** alebo **smerníka** (pointer *)
 - nie je možné použiť pre referenčné typy, pred použitím potrebné inicializovať napr. cez `Span<T>.Clear()`

```
int length = 3;
Span<int> numbers = stackalloc int[length];
for (var i = 0; i < length; i++)
{
    numbers[i] = i;
}
```

```
unsafe // Nebezpečný kód - použitie smerníka
{
    int length = 3;
    int* numbers = stackalloc int[length];
    for (var i = 0; i < length; i++)
    {
        numbers[i] = i;
    }
}
```

```
const int MaxStackLimit = 1024;
Span<byte> buffer;

if (inputLength <= MaxStackLimit)
{
    buffer = stackalloc byte[MaxStackLimit];
}
else
{
    buffer = new byte[inputLength];
}
```



Ako sa dostaneme k poslednému prvku poľa numbers?

① Start presenting to display the poll results on this slide.



Ako získame prvky v rozsahu od 0. indexu do 4. indexu vrátane?

① Start presenting to display the poll results on this slide.

slido

Please download and install the Slido app on all computers you use



Metóda Clone v poli vytvárá:

① Start presenting to display the poll results on this slide.



Ako sa deklaruje štvordimenzionálne pole celých čísiel?

① Start presenting to display the poll results on this slide.

slido

Please download and install the Slido app on all computers you use



Audience Q&A

① Start presenting to display the audience questions on this slide.

Znaky a reťazce

Typ	Typ (C#)	Typ (.NET)	Počet bitov
Znak (UTF-16)	char	System. Char	16
Reťazec	string	System. String	Kolekcia 16-bitových znakov (UTF-16)

```
public readonly struct Char : IComparable<char>, IConvertible, IEquatable<char>, IParsable<char>, ISpanParsable<char>,
Utf8SpanParsable<char>, System.Numerics.IAdditionOperators<char,char,char>,
System.Numerics.IAdditiveIdentity<char,char>, System.Numerics.IBinaryInteger<char>,
System.Numerics.IBinaryNumber<char>, System.Numerics.IBitwiseOperators<char,char,char>,
System.Numerics.IComparisonOperators<char,char,bool>, System.Numerics.IDecrementOperators<char>,
System.Numerics.IDivisionOperators<char,char,char>, System.Numerics.IEqualityOperators<char,char,bool>,
System.Numerics.IIncrementOperators<char>, System.Numerics.IMinMaxValue<char>,
System.Numerics.IModulusOperators<char,char,char>, System.Numerics.IMultiplicativeIdentity<char,char>,
System.Numerics.IMultiplyOperators<char,char,char>, System.Numerics.INumber<char>, System.Numerics.INumberBase<char>,
System.Numerics.IShiftOperators<char,int,char>, System.Numerics.ISubtractionOperators<char,char,char>,
System.Numerics.IUnaryNegationOperators<char,char>, System.Numerics.IUnaryPlusOperators<char,char>,
System.Numerics.IUnsignedNumber<char>
```

```
public sealed class String : ICloneable, IComparable, IComparable<string>, IConvertible, IEquatable<string>,
IParsable<string>, ISpanParsable<string>, System.Collections.Generic.IEnumerable<char>
```

String – jednoduchá práca s textom

- Niektoré zaujímavé metódy triedy `System.String`:
 - **Contains()**, **StartsWith()**, **EndsWith()** – zistí, či sa v reťazci nachádza zadaný reťazec kdekoľvek / na jeho začiatku / na jeho konci
 - **IndexOf()**, **LastIndexOf()** – vráti index, na ktorom sa nachádza zadaný reťazec (prehľadáva od začiatku / od konca), inak vráti -1
 - **Substring()** – vráti časť reťazca podľa zadaných parametrov
 - **Trim()**, **TrimStart()**, **TrimEnd()** – odstráni biele alebo iné zadané znaky
 - **Replace()** – nahradí časti reťazca zadaným reťazcom alebo znakom
 - **Split()** – rozdelí reťazec podľa zadaných znakov do poľa reťazcov
 - **Normalize()** – normalizuje reťazec do normalizačnej formy C
 - **String.Join()** – vytvorí reťazec z kolekcie reťazcov oddelený zadaným znakom

- **Unicode štandard** definuje **univerzálne kódovanie znakov**
- Jeden znak môže byť vyjadrený jedným alebo viacerými číselnými hodnotami (tzv. **code point** – Unicode scalar value), ku ktorým prináleží **popis**
 - Príklad: znak **A** je prezentovaný číslom **U+0041** a má popis **LATIN CAPITAL LETTER A**
 - Príklad: znak **Á** môže byť reprezentovaný:
 - jedným code pointom **U+00C1** (**LATIN CAPITAL LETTER A WITH ACUTE**) alebo
 - reťazcom zloženým z 2 code pointov **U+0041** (**LATIN CAPITAL LETTER A**) a **U+0301** (**COMBINING ACUTE ACCENT**)
- Unicode nedefinuje ako sa majú znaky zobrazíť (na to sú fonty, ktoré vykresľujú tzv. „glyphy“)
- Unicode štandard verzie 12.0 obsahuje:
 - 137 929 znakov
 - 1 114 112 code pointov
- **UTF (Unicode Transformation Format)** – Unicode štandard poskytuje rôzne možnosti ako zakódovať code point do code unit, čo môže byť číslo 8-bitové (**UTF-8**), 16-bitové (**UTF-16**) alebo 32-bitové (**UTF-32**)



slido

Please download and install the Slido app on all computers you use



Audience Q&A

① Start presenting to display the audience questions on this slide.

Práca s reťazcami (1)

- Vytvorenie dočasných reťazcov

```
string s1 = "Hello";  
string s2 = "World";  
string s3 = s1 + " " + s2;
```

Práca s reťazcami (2) – StringBuilder

- Používa „buffer“
- Dynamicky meniteľná veľkosť

```
StringBuilder sb = new("the quick");  
sb.Append(' ');  
sb.Append("brown fox jumped over ");  
sb.Append("the lazy dogs 1234567890 times");  
  
string s = sb.ToString();  
Console.WriteLine(s);
```

```
the quick brown fox jumped over the lazy dogs 1234567890 times
```

Formátovanie reťazcov

- Formáty čísiel, dátumov a časov

```
DateTime day = new(2025, 2, 14);  
Console.WriteLine($"{day:D}");  
Console.WriteLine($"{day:d}");  
Console.WriteLine($"{day:dd.MM.yyyy}");
```

piatok 14. februára 2025
14. 2. 2025
14.02.2025

```
int i = 2477;  
Console.WriteLine($"{i:n} {i:e} {i:x} {i:c}");
```

2 477,00 2,477000e+003 9ad 2 477,00 €

```
double d = 3.1415;  
Console.WriteLine($"{d:###.###}");  
Console.WriteLine($"{d:000.000}");  
Console.WriteLine();
```

3,142
003,142

FormattableString

- Obsahuje vlastnosti **Format** a **ArgumentCount**, metódy **GetArguments()**, **GetArgument()**

```
int x = 3, y = 4;

FormattableString s = $"The result of {x} + {y} is {x + y}";

Console.WriteLine($"format: {s.Format}");

for (int i = 0; i < s.ArgumentCount; i++)
{
    Console.WriteLine($"argument {{{i}}} = {s.GetArgument(i)}");
}
```

```
format: The result of {0} + {1} is {2}
argument {0} = 3
argument {1} = 4
argument {2} = 7
```


Doslovné reťazce (verbatim strings) s prefixom @

- Predpona @ pred znakom reťazca
- Keď sa použije, nie sú potrebné žiadne „escape“ znaky

```
string s = @"a tab: \t, a carriage return: \r, a newline: \n";  
Console.WriteLine(s);
```

```
a tab: \t, a carriage return: \r, a newline: \n
```

```
string s = "a tab: \t, a carriage return: \r, a newline: \n";  
Console.WriteLine(s);
```

```
a tab: , a carriage return:  
, a newline:
```

- Používa sa napr. v cestách:

```
string path1 = @"C:\Program Files\Microsoft\Visual Studio";  
string path2 = "C:\\Program Files\\Microsoft\\Visual Studio\\";
```

Rozsahy (ranges) s reťazcami

- Operátor rozsahu (range operator) **..**
- Operátor klobúka (hat operator) **^**
 - Hodnota **vľavo** – uzavretý interval, **vpravo** – otvorený interval

```
string s = "The quick brown fox jumped over the lazy dogs down 1234567890 times";  
  
string the = s[..3]; // Rozsah definovaný od začiatku (0) po index 3 - 1 = 2 vrátane  
string quick = s[4..9]; // Rozsah definovaný od indexu 4 po index 9 - 1 = 8 vrátane  
string times = s[^5..^0]; // Rozsah definovaný od indexu Length - 5 po index Length - 1  
  
Console.WriteLine(the);  
Console.WriteLine(quick);  
Console.WriteLine(times);
```

The
quick
times

slido

Please download and install the Slido app on all computers you use



Audience Q&A

① Start presenting to display the audience questions on this slide.

Menné priestory (namespaces)

- Triedy a iné typy sú v .NET usporiadané v **menných priestoroch**
 - Príklady: System, System.Collections.Generic, Microsoft.EntityFrameworkCore
- Odporúčanie pre pomenovanie ([guideline](#)) je v nasledovnej štruktúre:
<Company>.(<Product> | <Technology>)[.<Feature>][.<Subnamespace>]

```
namespace Uniza.CSharp.Lecture
{
    public class MyClass
    {
    }
}
```

```
namespace Uniza.CSharp.Lecture;

public class MyClass
{
}
```

```
// Plne kvalifikované meno triedy je Uniza.CSharp.Lecture.MyClass
var myObject = new Uniza.CSharp.Lecture.MyClass();
```

```
using Uniza.CSharp.Lecture;
var myObject = new MyClass();
```

```
namespace Uniza
{
    namespace CSharp
    {
        namespace Lecture
        {
            public class MyClass
            {
            }
        }
    }
}
```

Direktíva using (using directive)

- **Importovanie** menného priestoru:

```
using Uniza.CSharp.Lecture;  
var sample = new MyClass();
```

- Modikátor **global** (od C# 10) – using pre všetky súbory v kompilácii:

```
global using Uniza.CSharp.Lecture; // Zvyčajne sa definuje v jednom súbore v projekte  
  
var sample = new MyClass(); // Následne platné v akomkoľvek súbore v projekte
```

- **Alias** menného priestoru (**using VlastnyNazov = ExistujuciMennýPriestor;**):

```
using TimersTimer = System.Timers.Timer;  
using WebTimer = System.Web.UI.Timer;
```

- **Using static** – importuje statické členy z inej triedy:

```
using static System.Console;  
WriteLine("Hello, World!"); // Nie je potrebné písať Console.WriteLine
```

slido

Please download and install the Slido app on all computers you use



Audience Q&A

① Start presenting to display the audience questions on this slide.

Argumenty príkazového riadka (command-line arguments)

- Pre získanie **argumentov** z príkazového riadka použijeme **parameter** typu jednorozmerného poľa reťazcov (**string[]**) s ľubovoľne nazvaným identifikátorom, najčastejšie **args**:

```
class Program
{
    // Ak nepotrebujeme získať argumenty,
    // môže byť metóda Main aj bezparametrická.
    // Ak ale potrebujeme spracovať argumenty,
    // môžeme si parameter ľubovoľne pomenovať:
    static void Main(string[] args)
    {
        foreach (var argument in args)
        {
            Console.WriteLine(argument);
        }
    }
}
```

```
// Pole args je dostupné aj
// z „top-level statements“,
// hoci ho nevidíme:
foreach (var argument in args)
{
    Console.WriteLine(argument);
}
```

Terminológia – parameter vs. argument

- **Parameter** je premenná, ktorá prijíma hodnotu
 - V prípade metódy `Main(string[] args)` je to `string[] args`
- **Argument** je aktuálna hodnota, ktorú odovzdávame parametru
 - V prípade volania metódy v C# by argumentom mohla byť inštancia poľa, napr.:
`Main(new string[] { "hodnota1", "hodnota2", "hodnota3" })`
 - V prípade príkazového riadka predstavujú argumenty hodnoty, ktoré zadáme programu, napr.:
`dotnet run -- "hodnota1" "hodnota2" "hodnota3"` (cez .NET CLI)
`Program.exe "hodnota1" "hodnota2" "hodnota3"` (na Windowse)
`./Program "hodnota1" "hodnota2" "hodnota3"` (na Linuxe / MacOS)

VS Code – Main(string[] args) – pole argumentov (1)

Metóda **Main()** obsahuje parameter **args** predstavujúci pole argumentov, ktoré boli zadané aplikácii pri spustení. Cez **Length** zistíme veľkosť poľa (počet argumentov).

Cez **foreach** môžeme prejsť všetky položky poľa **args**.

Zadané **argumenty** sú **oddelené medzerami**. Ak chceme zadať iba **jeden argument** obsahujúci medzeru, musíme použiť **úvodzovky**, t. j.:
dotnet run -- "C# jazyk a .NET"

Cez indexer **args[číslo]** prečítame hodnotu reťazca z poľa **args**.

Argumenty, ktoré chceme dostať do poľa **args**, zadáme cez **dotnet run** **po voľbe --**

```
Program.cs
2 Console.WriteLine("Hello, World!");
3
4 if (args.Length > 0)
5 {
6     Console.WriteLine($"Ahoj, toto je vstup z parametra args[0]: {args[0]}!");
7     Console.WriteLine("Všetky argumenty:");
8     foreach (var argument in args)
9     {
10         Console.WriteLine(argument);
11     }
12 }
```

Terminal output:

```
PS C:\CSharp\Lecture01\HelloWorld> dotnet run -- C# jazyk a .NET
Ahoj, toto je vstup z parametra args[0]: C#!
Všetky argumenty:
C#
jazyk
a
.NET
PS C:\CSharp\Lecture01\HelloWorld>
```

VS Code – Main(string[] args) – pole argumentov (2)

The screenshot shows the Visual Studio Code interface with a C# file named `Program.cs` open. The code in the editor is as follows:

```
2 Console.WriteLine("Hello, World!");
3
4 if (args.Length > 0)
5 {
6     Console.WriteLine($"Ahoj, toto je vstup z parametra args[0]: {args[0]}!");
7     Console.WriteLine("Vsetky argumenty:");
8     foreach (var argument in args)
9     {
```

The bottom panel shows the `TERMINAL` tab with the following output:

```
a
.NET
PS C:\CSharp\Lecture01\HelloWorld> dotnet run -- Algebra "Jazyk C# a .NET"
Hello, World!
Ahoj, toto je vstup z parametra args[0]: Algebra!
Vsetky argumenty:
Algebra
Jazyk C# a .NET
PS C:\CSharp\Lecture01\HelloWorld>
```

A callout box points to the terminal output with the text: "Tu boli použité 2 argumenty vďaka použitiu úvodzoviek v druhom prípade (prvý **Algebra**, druhý **Jazyk C# a .NET**)".

VS Code – Main(string[] args) – pole argumentov (3)

The screenshot shows the Visual Studio Code interface with a C# program named `Program.cs` open. The program's logic is as follows:

```
2 Console.WriteLine("Hello, World!");
3
4 if (args.Length > 0)
5 {
6     Console.WriteLine($"Ahoj, toto je vstup z parametra args[0]: {args[0]}");
7     Console.WriteLine("Všetky argumenty:");
8     foreach (var argument in args)
9     {
10         Console.WriteLine(argument);
11     }
12 }
```

The `TERMINAL` panel at the bottom displays the execution output:

```
PS C:\CSharp\Lecture01\HelloWorld> cd .\bin\Debug\net7.0\
PS C:\CSharp\Lecture01\HelloWorld\bin\Debug\net7.0> .\HelloWorld.exe Algebra "Informatika 2" "Jazyk C# a .NET"
Hello, World!
Ahoj, toto je vstup z parametra args[0]: Algebra!
Všetky argumenty:
Algebra
Informatika 2
Jazyk C# a .NET
PS C:\CSharp\Lecture01\HelloWorld\bin\Debug\net7.0>
```

A yellow callout box on the right side of the image contains the following text:

Samozrejmosťou je spustenie programu EXE s argumentami, v tomto prípade sa **nepoužíva --**, ale **priamo sa argumenty zadávajú za názov** spúšťaného programu (v tomto prípade boli zadané 3 argumenty)

VS Code – Main(string[] args) – pole argumentov (4)

- Aby ste nemuseli zadávať tie isté argumenty cez príkazový riadok, môžete si vytvoriť priečinok **Properties** a do neho súbor **launchSettings.json** s týmto obsahom:

```
{
  "profiles": {
    "HelloWorld": {
      "commandName": "Project",
      "commandLineArgs": "Algebra \"Informatika 2\" \"Jazyk C# a .NET\"",
    }
  }
}
```

Názov profilu – ľubovoľný názov (napr. názov projektu – HelloWorld)

Nahradzte za vlastné argumenty – pozor na „escapovanie“ znaku úvodzovky (\\)

- Následne po spustení cez F5 alebo pomocou príkazu `dotnet run` sa tieto argumenty automaticky priradia aplikácii

VS Code – Main(string[] args) – pole argumentov (5)

V starších verziách VS Code sa argumenty definovali v súbore **launch.json**, dnes sa to už veľmi nepožíva

The screenshot shows the Visual Studio Code interface. The top editor pane displays the `launch.json` file for a project named "HelloWorld". The file is a JSON configuration for a ".NET Core Launch (console)" target. The `args` array is highlighted in blue, containing the values `"Algebra"`, `"Informatika 2"`, and `"Jazyk C# a .NET"`. A blue button labeled "Add Configuration..." is visible in the bottom right of the editor area. The bottom editor pane shows the terminal output of the program execution. The prompt is `PS C:\CSharp\Lecture01\HelloWorld>`. The command executed is `& 'c:\Users\stevk\.vscode\extensions\ms-dotnettools.csharp-1.25.4-win32-x64\.debugger\vsdbg.exe' '--interpreter=vscode' '--connection=e88c95f450c249fbb1d4371e63a831ea'`. The output shows "Hello, World!" followed by "Ahoj, toto je vstup z parametra args[0]: Algebra" and "Vsetky argumenty: Algebra Informatika 2 Jazyk C# a .NET". The status bar at the bottom indicates the active configuration is ".NET Core Launch (console) (HelloWorld)" and the current file is `launch.json` at line 14, column 69.

```
.vscode > {} launch.json > Launch Targets > {} .NET Core Launch (console)
13      "program": "${workspaceFolder}/bin/Debug/net7.0/HelloWorld.dll",
14      "args": ["Algebra", "Informatika 2", "Jazyk C# a .NET"],
15      "cwd": "${workspaceFolder}",
16      // For more information about the 'console' field, see https://aka.ms/VSCode-CS-Launch
17      "console": "integratedTerminal",
18      "internalConsoleOptions": "neverOpen",
19      "stopAtEntry": false
20  }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL POLYGLOT NOTEBOOK

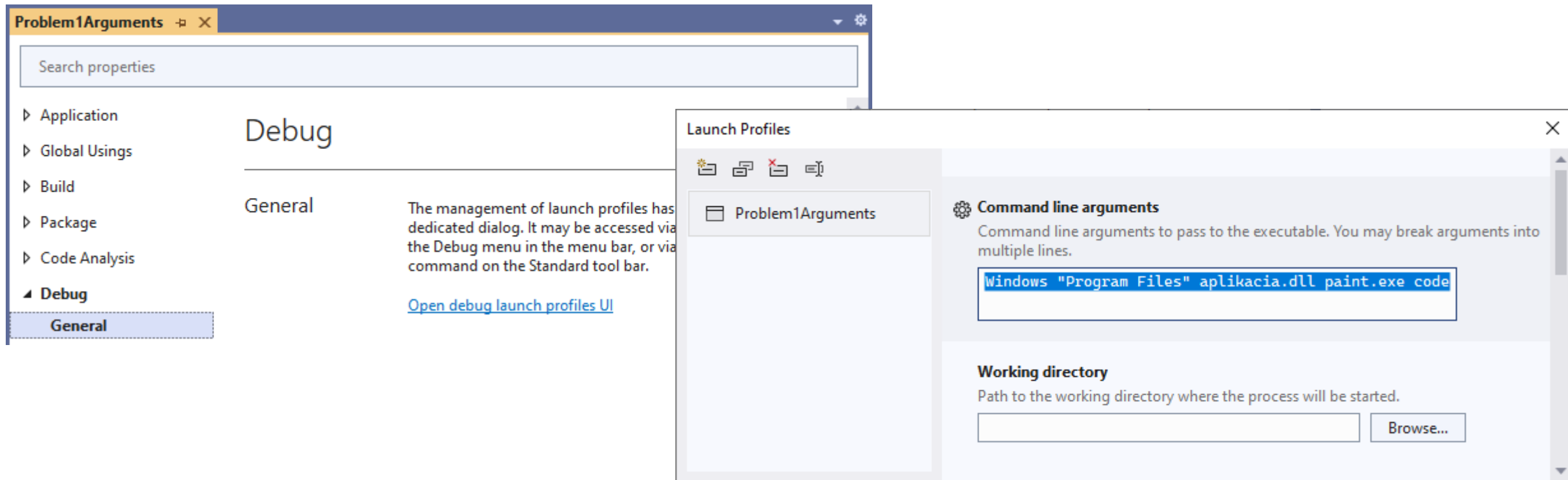
PS C:\CSharp\Lecture01\HelloWorld> & 'c:\Users\stevk\.vscode\extensions\ms-dotnettools.csharp-1.25.4-win32-x64\.debugger\vsdbg.exe' '--interpreter=vscode' '--connection=e88c95f450c249fbb1d4371e63a831ea'

Hello, World!
Ahoj, toto je vstup z parametra args[0]: Algebra
Vsetky argumenty:
Algebra
Informatika 2
Jazyk C# a .NET
PS C:\CSharp\Lecture01\HelloWorld>

Ln 14, Col 69 (56 selected) Spaces: 4 UTF-8 LF {} JSON with Comments

Visual Studio (1) – argumenty na ladenie

- Vo Visual Studiu môžeme definovať argumenty na ladenie:
 - Nastavenie projektu – **Properties** (Alt+Enter) na projekt -> **Debug** -> **Open debug launch profiles UI** -> **Command line arguments**:



Visual Studio (2) – argumenty na ladenie

- Vytvorí sa nám automaticky súbor **launchSettings.json**, ktorý môžeme upravovať buď priamo v tomto súbore alebo cez nastavenia projektu, ako v predchádzajúcom kroku:



```
1 {
2   "profiles": {
3     "Problem1Arguments": {
4       "commandName": "Project",
5       "commandLineArgs": "Windows \"Program Files\" aplikacia.dll paint.exe code"
6     }
7   }
8 }
```

- Poznámka: Tento súbor využíva aj .NET CLI príkaz **dotnet run**, preto ak ho spustíte, automaticky zoberie argumenty z tohto súboru
 - Ak by sme chceli spúšťať aplikáciu bez tohto súboru, použite príkaz:
dotnet run --no-launch-profile



**Ak zavoláme metódu
Console.WriteLine(text), čo
predstavuje platná premenná
text?**

① Start presenting to display the poll results on this slide.



Ako zadáme argumenty príkazového riadka cez .NET CLI, ak sa nachádzame v priečinku projektu?

① Start presenting to display the poll results on this slide.

slido

Please download and install the Slido app on all computers you use

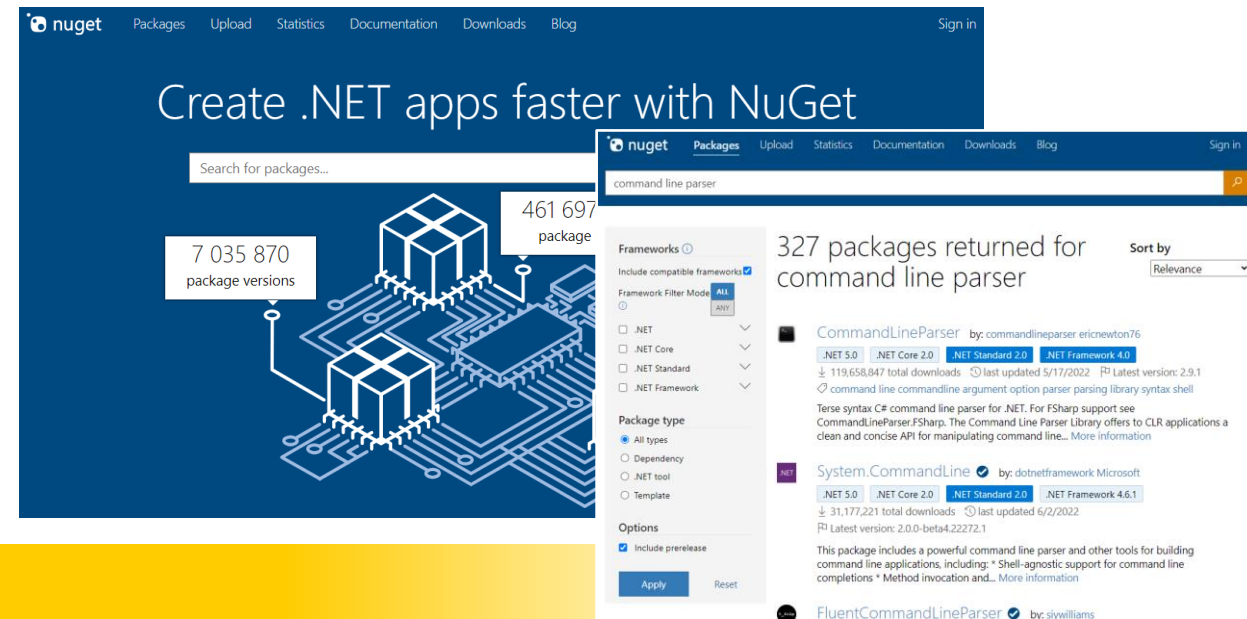


Audience Q&A

① Start presenting to display the audience questions on this slide.

NuGet (1)

- Je oficiálny **správca balíkov pre .NET**, navrhnutý na zdieľanie znovupoužiteľného kódu vo forme balíčkov, umožňuje tak rýchlejšie vytvárať .NET aplikácie zdieľaním a využívaním užitočných knižníc
- **NuGet balíček** je **ZIP súbor** s príponou **.nupkg**, obsahujúci jedno alebo viacero zostavení (skompilovaných DLL súborov), spolu s ďalšími súvisiacimi súbormi a konfiguračnými informáciami
- **Verejné balíčky** sú jednoducho **prístupné cez www.nuget.org**, okrem toho je možné hostiť balíčky súkromne na klaude, privátnej sieti alebo lokálne



NuGet (2)

- NuGet balíčky je možné do aplikácie pridať pomocou:
 - .NET CLI príkazu:
 - **dotnet add package** **NázovBalíčka**
 - Visual Studio pre Windows:
 - cez **NuGet Package Manager** okno vybratím príkazu **Manage NuGet Packages** alebo
 - cez **PowerShell** príkaz z **Package Manager Console**: **Install-Package** **NázovBalíčka**
 - Manuálnou úpravou .csproj súboru:
 - pridaním **<PackageReference Include="NázovBalíčka" Version="VerziaBalíčka" />**
- Okrem pridania sú k dispozícii príkazy aj na aktualizáciu, vyhľadanie a odobratie, viac informácií v dokumentácii:
<https://learn.microsoft.com/en-us/nuget/>

NuGet (3) – príklad balíčka Figgle – .NET CLI

- Na pridanie jednoduchého balíčka [Figgle](#) (na generovanie ASCII bannerov) môžeme použiť .NET CLI príkaz:

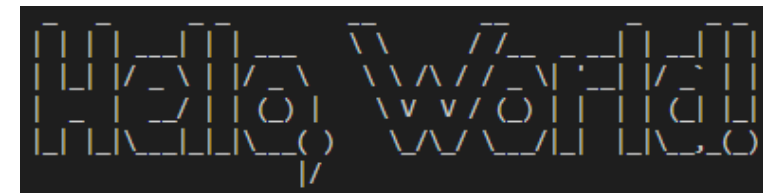
dotnet add package Figgle

- Do projektu sa pridá najnovšia verzia balíčka Figgle, projektový súbor **.csproj** sa upraví nasledovne:

```
<Project Sdk="Microsoft.NET.Sdk">
  ...
  <ItemGroup>
    <PackageReference Include="Figgle" Version="0.5.1" />
  </ItemGroup>
</Project>
```

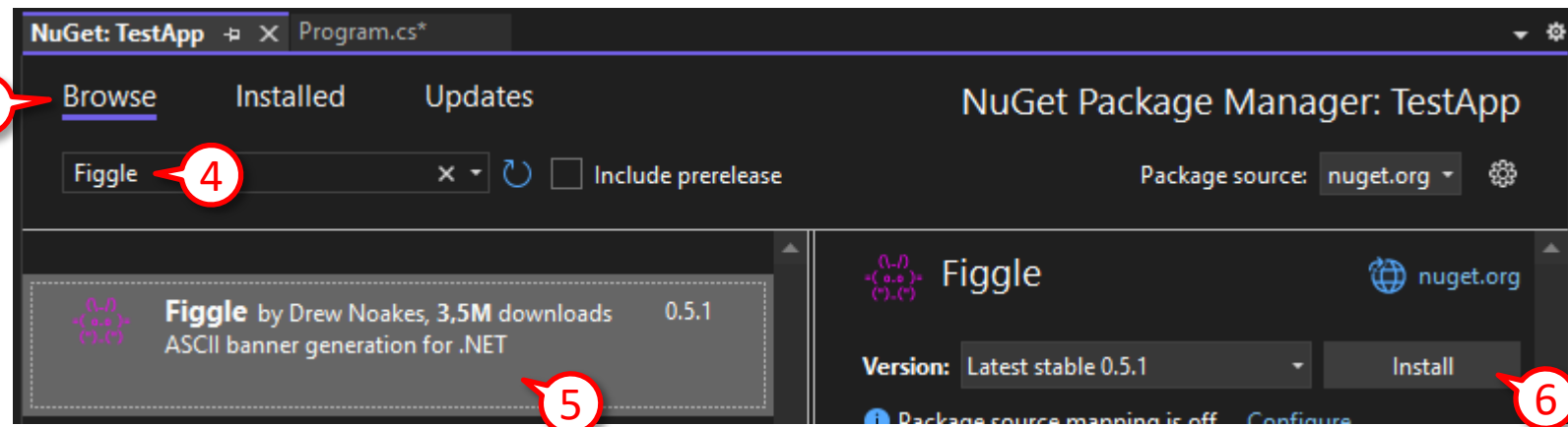
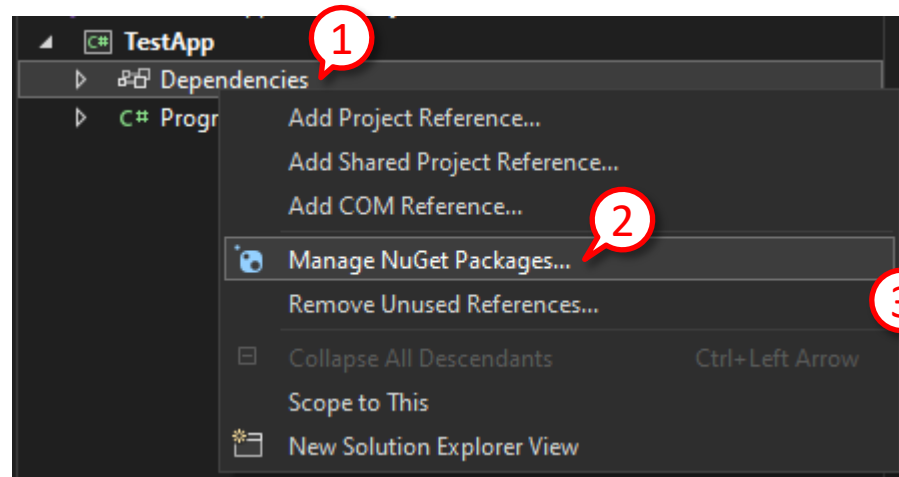
- Následne môžeme knižnicu použiť napr. takto:

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



NuGet (4) – príklad balíčka Figgle – Visual Studio

- Namiesto .NET CLI príkazu môžete pridať balíček vo Visual Studiu vo Windows „klikacím“ spôsobom nasledovne:
 - Kliknite v projekte pravým tlačidlom na **Dependencies** a v kontextovom menu vyberte **Manage NuGet Packages...**, čím sa otvorí okno **NuGet Package Manager**, v ktorom vyhľadajte a nainštalujte požadovaný balíček



NuGet (5) – príklad balíčka System.CommandLine

- Další užitočný balíček je [System.CommandLine](#), ktorý umožňuje jednoducho spracovať argumenty z príkazového riadka
 - Pridanie: **dotnet add package System.CommandLine --prerelease** (keďže stále je v beta verzii, musíme použiť voľbu --prerelease; vo Visual Studiu musíme začiarknuť políčko „Include prerelease“ pred vyhľadáním v NuGet Package Manager okne)
- Alebo [System.CommandLine.DragonFruit](#):
 - Pridanie: **dotnet add package System.CommandLine.DragonFruit --prerelease**
- Viac informácií:
 - <https://learn.microsoft.com/en-us/dotnet/standard/commandline/>
 - <https://github.com/dotnet/command-line-api/blob/main/docs/DragonFruit-overview.md>
 - https://www.youtube.com/watch?v=kM0pWAwo_FQ

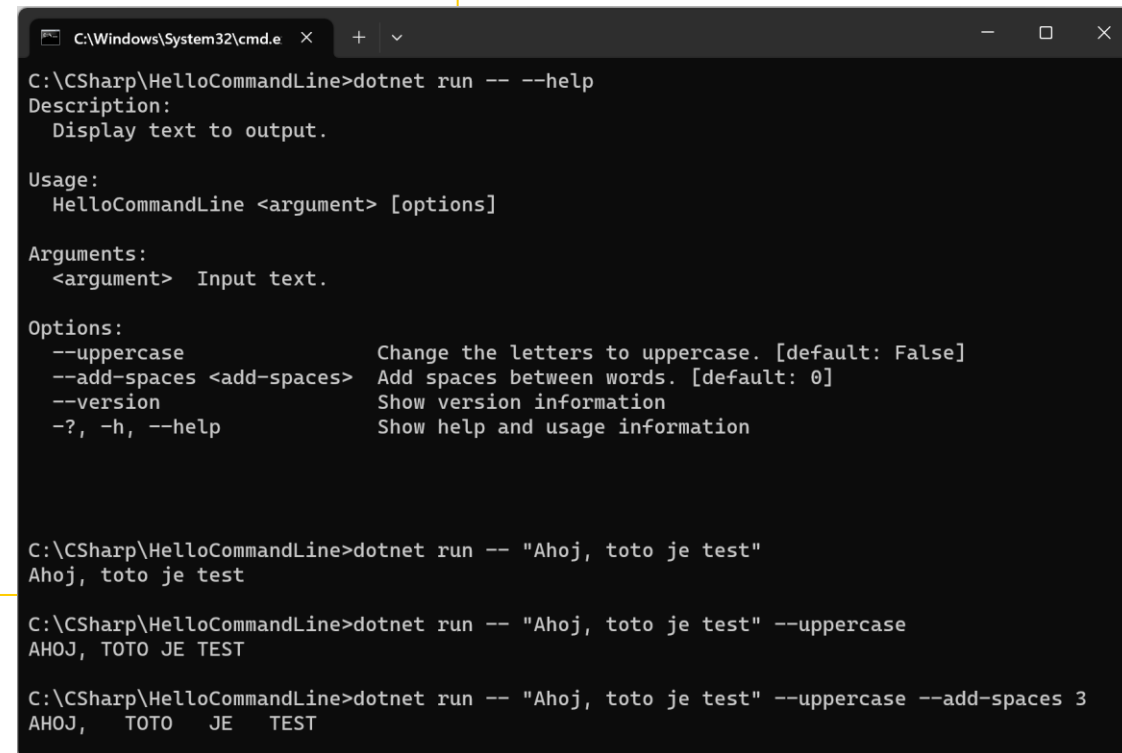
NuGet (6) – System.CommandLine.DragonFruit

Ak chceme mať „argument“ a nie „option“ v DragonFruit, musíme nazvať parameter v Main metóde nasledovne:
args, argument alebo arguments

```
class Program
{
    /// <summary>
    /// Display text to output.
    /// </summary>
    /// <param name="argument">Input text.</param>
    /// <param name="uppercase">Change the letters to uppercase.</param>
    /// <param name="addSpaces">Add spaces between words.</param>
    private static void Main(string argument, bool uppercase = false, int addSpaces = 0)
    {
        if (uppercase)
            argument = argument.ToUpper();

        if (addSpaces > 0)
        {
            var newSpaces = new string(' ', addSpaces);
            argument = argument.Replace(" ", newSpaces);
        }

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



```
C:\Windows\System32\cmd.exe
C:\CSharp\HelloCommandLine>dotnet run -- --help
Description:
  Display text to output.

Usage:
  HelloCommandLine <argument> [options]

Arguments:
  <argument>  Input text.

Options:
  --uppercase           Change the letters to uppercase. [default: False]
  --add-spaces <add-spaces> Add spaces between words. [default: 0]
  --version            Show version information
  -?, -h, --help      Show help and usage information

C:\CSharp\HelloCommandLine>dotnet run -- "Ahoj, toto je test"
Ahoj, toto je test

C:\CSharp\HelloCommandLine>dotnet run -- "Ahoj, toto je test" --uppercase
AHOJ, TOTO JE TEST

C:\CSharp\HelloCommandLine>dotnet run -- "Ahoj, toto je test" --uppercase --add-spaces 3
AHOJ,   TOTO   JE   TEST
```


NuGet (7) – System.CommandLine

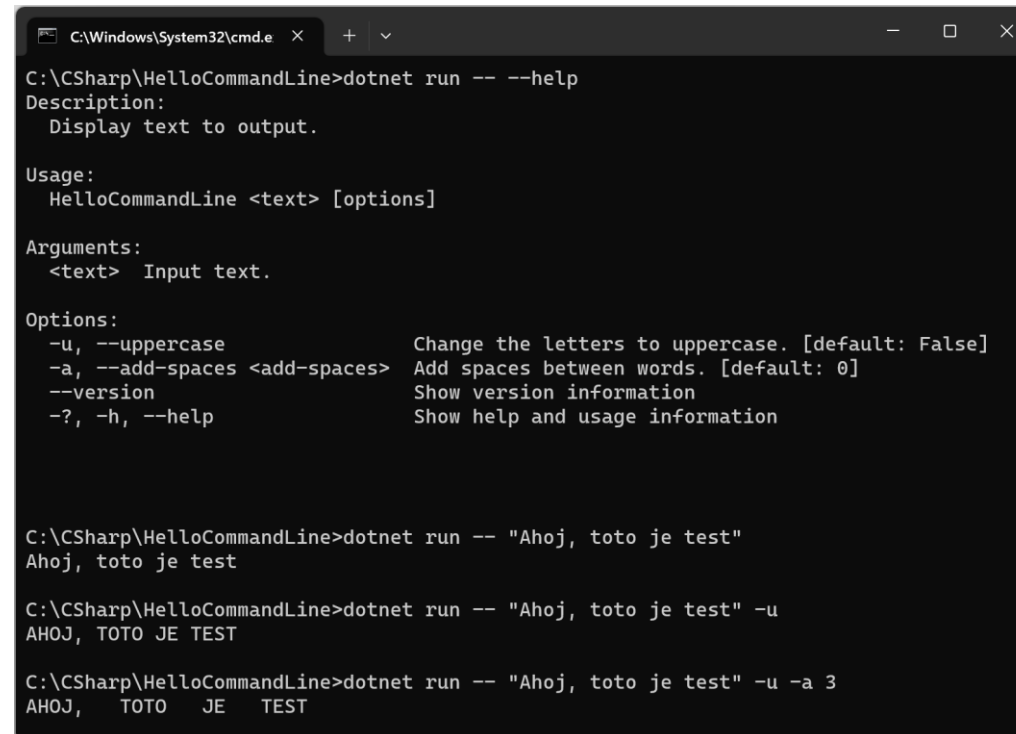
```
using System.CommandLine;

// Deklarujeme si argumenty, príkazy a voľby, ktoré chceme spracovávať
var textArgument = new Argument<string>("text", "Input text.");
var uppercaseOption = new Option<bool>(new[] { "-u", "--uppercase" }, () => false, "Change the letters to uppercase.");
var addSpacesOption = new Option<int>(new[] { "-a", "--add-spaces" }, () => 0, "Add spaces between words.");
var rootCommand = new RootCommand("Display text to output.") { textArgument, uppercaseOption, addSpacesOption };
rootCommand.SetHandler((text, uppercase, addSpaces) => DisplayToOutput(text, uppercase, addSpaces),
    textArgument, uppercaseOption, addSpacesOption);
return rootCommand.Invoke(args);

// V metóde spracujeme argumenty, ktoré nám knižnica rozparsuje:
void DisplayToOutput(string text, bool uppercase, int addSpaces)
{
    if (uppercase)
        text = text.ToUpper();

    if (addSpaces > 0)
    {
        var newSpaces = new string(' ', addSpaces);
        text = text.Replace(" ", newSpaces);
    }

    Console.WriteLine(text);
}
```



```
C:\Windows\System32\cmd.exe
C:\CS\HelloCommandLine>dotnet run -- --help
Description:
  Display text to output.

Usage:
  HelloCommandLine <text> [options]

Arguments:
  <text>  Input text.

Options:
  -u, --uppercase          Change the letters to uppercase. [default: False]
  -a, --add-spaces <add-spaces> Add spaces between words. [default: 0]
  --version                Show version information
  -?, -h, --help           Show help and usage information

C:\CS\HelloCommandLine>dotnet run -- "Ahoj, toto je test"
Ahoj, toto je test

C:\CS\HelloCommandLine>dotnet run -- "Ahoj, toto je test" -u
AHOJ, TOTO JE TEST

C:\CS\HelloCommandLine>dotnet run -- "Ahoj, toto je test" -u -a 3
AHOJ,   TOTO   JE   TEST
```

NuGet (8) – CommandLineParser (1)

```
using CommandLine;

class Program
{
    public class Options
    {
        [Option('a', "argument", Required = true, HelpText = "Input text.")]
        public string Argument { get; set; }

        [Option('u', "uppercase", Default = false, HelpText = "Change the letters to uppercase.")]
        public bool Uppercase { get; set; }

        [Option('s', "addSpaces", Default = 0, HelpText = "Add spaces between words.")]
        public int AddSpaces { get; set; }
    }

    static void Main(string[] args)
    {
        Parser.Default.ParseArguments<Options>(args)
            .WithParsed<Options>(options => RunOptions(options))
            .WithNotParsed<Options>((errors) => HandleParseError(errors));
    }

    // ...
}
```

DOPLNIŤ SCREENSHOTY

NuGet (9) – CommandLineParser (2)

```
// ...

private static void RunOptions(Options options)
{
    string argument = options.Argument;

    if (options.Uppercase)
        argument = argument.ToUpper();

    if (options.AddSpaces > 0)
    {
        var newSpaces = new string(' ', options.AddSpaces);
        argument = argument.Replace(" ", newSpaces);
    }

    Console.WriteLine(argument);
}

private static void HandleParseError(IEnumerable<Error> errors)
{
    // Dodatočné spracovanie chýb
}
}
```

- Inou alternatívou je použitie balíčka [Microsoft.Extensions.Configuration.CommandLine](#) z rodiny [konfiguračných](#) balíčkov Microsoft.Extensions.Configuration.*
- Zadať **argument** je následne možné s **medzerou** za **parametrom** povinne s prefixom **--** alebo **/**, alebo nepovinne s **=** (vtedy nie je potrebné zadávať prefix):

```
--output "C:\OutputData"  
/output "C:\OutputData"  
output="C:\OutputData"
```

```
--output="C:\OutputData"  
/output="C:\OutputData"
```

```
static void Main(string[] args)  
{  
    IConfiguration configuration = new ConfigurationBuilder()  
        .AddCommandLine(args)  
        .Build();  
  
    Console.WriteLine($"InputPath: {configuration["input"]}");  
    Console.WriteLine($"OutputPath: {configuration["output"]}");  
}
```

```
dotnet run --input "C:\InputData" --output "C:\OutputData"
```

slido

Please download and install the Slido app on all computers you use



Audience Q&A

① Start presenting to display the audience questions on this slide.

Použitá literatúra

- Christian Nagel: **Professional C# and .NET** 2021 Edition, 8. vydanie, ISBN-13: 978-1119797203
- .NET a C#:
 - <https://learn.microsoft.com/en-us/dotnet/>
 - <https://learn.microsoft.com/en-us/dotnet/csharp/>



ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Katedra softvérových
technológií

stefan.toth@uniza.sk

ĎAKUJEM ZA POZORNOSŤ

Upozornenie

- Tieto študijné materiály sú určené výhradne pre študentov predmetu **Jazyk C# a .NET** na Fakulte riadenia a informatiky Žilinskej univerzity v Žiline
- Reprodukovanie, šírenie (i častí) materiálov bez písomného súhlasu autora nie je dovolené



ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Ing. **Štefan Toth**, PhD.
stefan.toth@uniza.sk