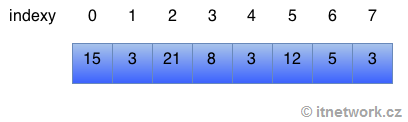
Pole v C#

Pokud potřebujeme uchovávat **větší množství proměnných stejného typu,** použijte datový typ pole. Můžeme si ho představit jako řadu přihrádek, kde v každé máme uložený jeden prvek. Přihrádky jsou očíslované tzv. indexy, první má index 0.



*(Na obrázku je vidět pole osmi čísel)*

Programovací jazyky se velmi liší v tom, jak s polem pracují. V některých jazycích (zejména starších, kompilovaných) nebylo možné za běhu programu vytvořit pole s dynamickou velikostí (např. mu dát velikost dle nějaké proměnné). Pole se muselo deklarovat s konstantní velikostí přímo ve zdrojovém kódu v C++. C# je virtuální stroj, tedy cosi mezi kompilerem a interpretem. Proto můžeme pole založit s velikostí, kterou dynamicky zadáme až za běhu programu, ale velikost existujícího pole modifikovat nemůžeme.

Možná vás napadá, proč se tu zabýváme s polem, když má evidentně mnoho omezení a existují lepší datové struktury. Odpověď je prostá: pole je totiž jednoduché. Nemyslím pro nás na pochopení (to také), ale zejména pro C#. Rychle se s ním pracuje, protože prvky jsou v paměti jednoduše uloženy za sebou, zabírají všechny stejně místa a rychle se k nim přistupuje. Mnoho vnitřních funkčností v .NET proto nějak pracuje s polem nebo pole vrací. Je to klíčová struktura.

Pole je objektem referenčního typu, který je potomkem třídy System.Array.

Deklarace: datový\_typ[] identifikátor;

Pro hromadnou manipulaci s prvky pole se používají cykly.

Pole deklarujeme pomocí hranatých závorek:

**int**[] pole;

*pole* je samozřejmě název naší proměnné. Nyní jsme však pouze deklarovali, že v proměnné bude pole intů.

Tímto je vytvořena referenční proměnná typu pole, která má hodnotu null, instance objektu ještě není vytvořena.

Nyní ho musíme založit, abychom ho mohli používat. Použijeme k tomu klíčové slovo **new**, které zatím nebudeme vysvětlovat. Spokojme se s tím, že je to kvůli tomu, že je pole referenční datový typ (můžeme chápat jako složitější typ):

**int**[] pole = **new** **int**[10];

Nyní máme v proměnné *pole* pole o velikosti deseti intů. Všech 10 prvků v poli má hodnotu 0.

K prvkům pole potom přistupujeme přes hranatou závorku, pojďme na první index (tedy index 0) uložit číslo 1.

**int**[] pole = **new** **int**[10];

pole[0] = 1;

Pole je možno inicializovat hned při deklaraci

**int**[] pole = **new** **int**[4]{1,2,3,4};

**int**[] pole = {1,2,3,4};//zjednodušený zápis

**string**[] simpsonovi = {"Homer", "Marge", "Bart", "Lisa", "Meggie"};

Plnit pole takhle ručně by bylo příliš pracné, použijeme cyklus a naplníme si pole čísly od 1 do 10. K naplnění použijeme for cyklus:

**int**[] pole = **new** **int**[10];

**for** (**int** i = 0; i < 10; i++)

pole[i] = i + 1;

Abychom pole vypsali, můžeme za předchozí kód připsat:

**for** (**int** i = 0; i < pole.Length; i++)

listBox1.Items.Add(pole[i].ToString());

Všimněte si, že pole má vlastnost Length, kde je uložena jeho délka, tedy počet prvků.

Můžeme použít zjednodušenou verzi cyklu pro práci s kolekcemi, známou jako foreach. Ten projede všechny prvky v poli a jeho délku si zjistí sám. Jeho syntaxe je následující:

**foreach** (datovytyp promenna **in** kolekce)

{

// příkazy

}

Cyklus projede prvky v kolekci (to je obecný název pro struktury, které obsahují více prvků, u nás to bude pole) postupně od prvního do posledního. Prvek máme v každé iteraci cyklu uložený v dané proměnné.

Přepišme tedy náš dosavadní program pro foreach. **Proměnná pole** je pouze pro **čtení**, cyklus foreach nelze použít pro načtení hodnot do pole, použijeme ho jen pro výpis pole. Cyklus foreach **nemá řídící proměnnou**.

**int**[] pole = **new** **int**[10];

**for** (**int** i = 0; i < 10; i++)

pole[i] = i + 1;

**foreach** (**int** prvek **in** pole)

listBox1.Items.Add(prvek.ToString());

Výstup programu:

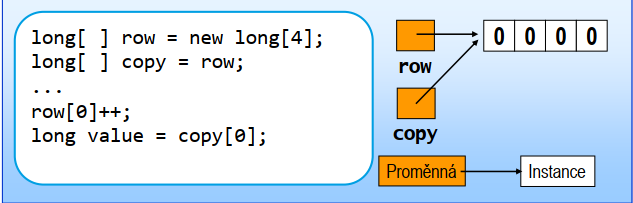
1 2 3 4 5 6 7 8 9 10

### Kopírování proměnné typu pole

Pole můžeme vytvořit i přiřazením pole s prvky stejného druhu

Přiřazení proměnné typu pole do jiné, nevytvoří kopii pole

* Pole je vždy referenční typ, tzn. že proměnná pole je na haldě
* Dvě proměnné typu pole mohou ukazovat na to samé pole.



**int**[] pole = cisla;

//pozor, cokoliv uděláme v proměnné pole, stejné změny budou i v proměnné cisla

**string**[] poleStringu = textBox1.Lines;

//pole můžeme i jednoduše do textBox1 vypsat

textBox1.Lines = poleStringu;

**Metody na třídě Array**

.NET nám poskytuje třídu Array, která obsahuje pomocné metody pro práci s poli.

**Sort()**

Jak již název napovídá, metoda nám pole seřadí. Její jediný parametr je pole, které chceme seřadit. Je dokonce tak chytrá, že pracuje podle toho, co máme v poli uložené. Stringy třídí podle abecedy, čísla podle velikosti. Zkusme si setřídit a vypsat naši rodinku Simpsnů:

**string**[] simpsonovi = {"Homer", "Marge", "Bart", "Lisa", "Meggie"};

Array.Sort(simpsonovi);

**foreach** (**string** s **in** simpsonovi)

listBox1.Items.Add(s);

Výstup programu:

Bart

Homer

Lisa

Marge

Meggie

**Reverse()**

Reverse() nám pole otočí (první prvek bude jako poslední atd.), toho můžeme využít např. pro třídění pozpátku:

**string**[] simpsonovi = {"Homer", "Marge", "Bart", "Lisa", "Meggie"};

Array.Sort(simpsonovi);

Array.Reverse(simpsonovi);

**foreach** (**string** s **in** simpsonovi)

listBox1.Items.Add(s);

**IndexOf() a LastIndexOf()**

Tyto metody vrátí index prvního nebo posledního nalezeného prvku. V případě nenalezení prvku vrátí -1. Každá z metod bere dva parametry, prvním je pole, druhým hledaný prvek. Umožníme uživateli zadat jméno Simpsna a řekneme mu, kterého Simsna asi vybral.

**string**[] simpsonovi = {"Homer", "Marge", "Bart", "Lisa", "Meggie"};

//"Ahoj, zadej svého oblíbeného Simpsna (z rodiny Simpsů):

**string** simpson = textBox1.Text;

**int** pozice = Array.IndexOf(simpsonovi, simpson);

**if** (pozice >= 0)

MessageBox.Show("Jo, {0} je můj nejoblíbenější Simpson!", simpsonovi[pozice]);

**else**

MessageBox.Show ("Hele, tohle není Simpson!");

**Copy()**

Copy již podle názvu zkopíruje část pole do jiného pole. Prvním parametrem je zdrojové pole, druhým cílové a třetím počet prvků, který se má zkopírovat.

**Metody na poli**

Třída Array není jedinou možností, jak s polem manipulovat. Přímo na samotné instanci pole (konkrétní proměnné) můžeme volat také spoustu metod. I když si zmíníme jen některé, je jich opravdu hodně.

**Length**

public int Length { get; }

Třída System.Array, tedy i každé námi vytvořené pole obsahuje vlastnost **Length**, která je pouze **read-only** a je pomocí ní možné zjistit **celkovou** délku pole.   
Typ vlastnosti Length je **Int32**.

**int**[] pole = **new** **int**[4];

int delka = pole.Length;//je vracena hodnota 4

Length je vlastnost, ne metoda, nepíší se za ni tedy závorky ().

**Min(), Max(), Average(), Sum()**

Matematické metody, vracející nejmenší prvek (Min()), největší prvek (Max()), průměr ze všech prvků (Average()) a součet všech prvků (Sum()). Metody nemají žádné parametry.

int max = pole.Max();

**Concat(), Intersect(), Union()**

Všechny tyto metody vrátí na výstupu novou datový typ IEnumerable. **Concat()** vykoná nám již známou konkatenaci, tedy k našemu poli **připojí** druhé pole a takto vzniklé nové pole vrátí. **Intersect()** vykoná **průnik** obou polí, tedy sestaví pole s prvky, které jsou oběma polím společné. **Union()** naopak vykoná **sjednocení**, funguje tedy podobně jako Concat(), jen prvky, které byly v obou polích, jsou v novém poli jen jednou. Aby bylo dosazení výsledku zpět do pole možné, musíme ho ještě přetypovat na pole metodou ToArray().

int[] poleC = poleA.Concat(poleB).ToArray();

**First() a Last()**

Již podle názvu metody vrátí první a poslední prvek, nemají žádné parametry.

int prvni = pole.First();

**Take() a Skip()**

Obě tyto metody berou jako parametr počet prvků. Také vrátí pole s daným počtem prvků zkopírovaných od začátku původního pole. Skip naopak vrátí pole bez těchto prvních prvků.

pole = pole.Skip(5).ToArray();//pole bez 5 prvních prvků

pole = pole.Take(5).ToArray();//pole bude obsahovat 5 prvních prvků

**Contains()**

Metoda vrací true/false podle toho, zda se prvek, uvedený v parametru metody, v daném poli nachází.

if (pole.Contains(6)) //zjišťuje, zda se v poli nachází prvek s číselm 6

**Reverse()**

Metodu Reverse známe již z třídy Array, pokud ji ale voláme na konkrétním poli, tak se prvky v něm neotočí, nýbrž je vytvořeno nové otočené pole a to je vráceno. Metoda nemá žádné parametry.

pole = pole.Reverse().ToArray();//otočí prvky v poli

int[] pole2 = pole.Reverse().ToArray();//do nového pole překopíruje otočné prvky

**Distinct()**

Distinct je metoda bez parametrů a zajistí, aby byl v poli každý prvek jen jednou, tedy vymaže duplicitní prvky a unikátní pole vrátí jako návratovou hodnotu metody, opět tedy nemodifikuje dané pole.

pole = pole.Distinct().ToArray();//smaže v poli všechny duplicitní prvky

int[] pole2 = pole.Distinct().ToArray();//překopíruje do pole všechny jedinečné prvky

**int**[] cisla = { 1, 2, 3, 3, 3, 5, 5 };

cisla = cisla.Distinct().ToArray();//pole bude obsahovat1,2,3,5

**CopyTo()**

**int**[] pole2 = **new** **int**[10];pole.CopyTo(pole2,0);

Do pole2 se nakopírují od nultého indexu prvky z pole. Tyto dvě proměnné nejsou na sobě závislé. Při změně hodnot v proměnné pole nedojde ke změně hodnost v proměnné pole2

*Mnoho metod nemění přímo naše pole, ale vrátí pouze pole nové (jsou to metody Concat, Intersect, Union, Reverse a Distinct), ve kterém jsou provedeny požadované změny. Pokud chceme modifikovat původní pole, musíme do něj dosadit. Tyto metody bohužel z důvodů, které pochopíme až později, nevrací přímo pole, ale typ IEnumerable. Aby bylo dosazení výsledku zpět do pole možné, musíme ho ještě převést na pole metodou ToArray().*

Proměnná délka pole

Říkali jsme si, že délku pole můžeme definovat i za běhu programu.

**int** pocet = Convert.ToInt32(textBox1.Text);

**int**[] cisla = **new** **int**[pocet];

Shrnutí:

* Pole slouží k uchovávání většího množství proměnných stejného typu.
* Pole v C# můžeme založit s velikostí, kterou dynamicky zadáme až za běhu programu.
* Velikost již existujícího pole již nemůžeme modifikovat. (V případě této potřeba musíme vytvořit novou instanci)
* Prvky pole jsou v paměti jednoduše uloženy za sebou, zabírají všechny stejně místa a rychle se k nim přistupuje.  
  To je důvod proč používat pole, přestože má evidentně mnoho omezení a existují dokonalejší datové struktury

Příklady

1. Navrhněte program, který vytvoří jednorozměrné pole **n** náhodně vygenerovaných **reálných** čísel v intervalu <1;1000)a vyhledá **maximální** a **minimální** prvek pole. Nalezené maximum a minimum v poli **vyměňte**. Spočítejte **aritmetický průměr** všech čísel bez maximálního a minimálního prvku. Zobrazte původní i výsledné pole (čísla v poli zaokrouhlete na dvě desetinná místa (parametr u metody ToString("F2")))

Příklad vyřešte dvěma způsoby bez použití metod a s použitím metod.

1. Do pole načtěte **n** náhodných celých čísel z intervalu **1.. 10** včetně**.** Pole **setřiďte** podle velikosti. Pomoci RadioButtons vyberte zda, bude pole setříděné **sestupně** nebo **vzestupně**, Zobrazte původní i výsledné pole. Následně vypište **druhý** **největší** a **druhý** **nejmenší** prvek.
2. Načtěte pole A o N náhodných celých číslech z intervalu (2,10>, načtěte pole B o M náhodných celých číslech z intervalu (2,10>. Vytvořte pole C, které bude obsahovat **všechny** prvky z pole A i B. Vytvořte pole D, který je **sjednocením** pole A a B. Vytvořte pole E, který je **průnikem** pole A a B. Všechna pole vytvořte a zobrazte pomocí 5 tlačítek.
3. Do pole načtěte **n** náhodných celých čísel z intervalu 1..20 včetně. Z pole **vymažte** všechny **duplicitní** prvky, dále odstraňte minimum a maximum. Původní i upravené pole zobrazte.
4. Do pole načtěte N náhodných čísel (0,20). Pole zobrazte setříděné. Vytvořte nové pole, které bude obsahovat všechna čísla menší nebo rovno než zadané číslo B. Pole zobrazte. Vytvořte nové pole, které bude obsahovat všechna čísla větší než zadané číslo B. Použijte metodu **Sort, Skip** a **Take**.
5. Na vstupu je celé číslo n. Do pole vygenerujte n náhodných reálných čísel z intervalu <-1000;1000). Pole zobrazte do ListBoxu. Najděte maximum z těch prvků pole, které jsou menší nebo rovny aritmetickému průměru všech prvků pole. Pro hledání maxima použijte cyklus foreach.
6. V komponentě **ListBox** jsou nezáporná celá čísla, mezi nimi se vyskytuje alespoň minimálně jedna 0. **Kladná** čísla z ListBox až po první 0 uložte do pole. Vypište **podíl** **prvního** prvku a **posledního** prvku pole. **Vypište** všechny **prvky** včetně jejich **pořadí** v poli, které jsou **menší** než **poslední** prvek pole.
7. Do pole znaků vygenerujte náhodné znaky v rozsahu ‘0’..’B’ včetně. Pokud je **poslední** znak pole **cifra**, **nahraďte** všechny cifry v poli znakem **x**. V opačném případě najděte znak s **maximální ASCII hodnotou,** vyměňte jej s **posledním** prvkem pole. Opravené pole zobrazte.
8. Na vstupu je celé číslo n. Do pole vygenerujte n náhodných celých čísel z intervalu 1..10. Vypište hodnotu **mediánu** a **počet**, kolikrát se tato hodnota v poli **vyskytuje**.
9. V komponentě TextBox je zapsán řetězec znaků. Všechny znaky z řetězce uložte do pole znaků a pole zobrazte. Najděte znak pole, který se v poli vyskytuje v **největším** **počtu**. Tento znak vypište nalezený znak i s počtem jeho výskytů v poli