Pole a listy

Úvod

V této kapitole se budeme věnovat složitějším datovým strukturám `pole` (array) a `List`, které slouží pro vytvoření seznamu hodnot, přístupných v jedné proměnné pod určitými indexy.

Co je to pole a k čemu se používá?

`Pole` (array) je datová struktura umožňující udržování hodnot stejného datového typu pohromadě a ulehčující práci s daty. Pro ukázku praktičnosti si představme úlohu, kdy máme skupinu lidí, která má předem známý počet osob. Chtěli bychom si u každé osoby zaznamenat, jaký má měsíční příjem a poté zjistit průměrný příjem v této skupině. K této úloze se perfektně hodí právě pole. Pro jednoduchost předpokládejme, že všechny příjmy jsou celočíselné a naše skupina lidí obsahuje 5 osob.

Příklad

```
// Založení nového pole o délce 5
int[] prijmy = new int[5];

// Naplnění pole hodnotami
prijmy[0] = 20000;
prijmy[1] = 15000;
prijmy[2] = 25000;
prijmy[3] = 30000;
prijmy[4] = 12000;

// Výpočet průměru
double prumer = (prijmy[0] + prijmy[1] + prijmy[2] + prijmy[3] + prijmy[4]) / prijmy.Length;

// Výpis průměru
Console.WriteLine(prumer);
```

Výstupem tohoto kódu bude `20 400`, což je náš hledaný průměrný příjem. Pojďme si nyní projít náš kód. Pole deklarujeme pomocí hranatých závorek za datovým typem (v našem případě `int[]`). Dále následuje název samotného pole (`prijmy`) a poté operátor přiřazení `=`. Nakonec uvádíme

klíčové slovo `new` následované znovu zvoleným datovým typem s hranatými závorkami, ve kterých je maximální počet prvků, které chceme do pole uložit.

K jednotlivým prvkům v poli pak přistupujeme pomocí tzv. `indexů`. Pro pole o `n` prvcích jsou to celá čísla, počínající `0` a končící hodnotou `n-1`. Pro `5` prvků jsou to tedy čísla `0`, `1`, `2`, `3` a `4`. Pro výběr prvku na určité pozici slouží hranaté závorky, uvnitř kterých uvádíme samotný index. Pro přiřazení na zvolený index slouží operátor přiřazení (`=`). Pro přiřazení hodnoty `20 000` na první pozici (index `0`) tedy použijeme příkaz `prijmy[0] = 20000`. Pro vypsání této hodnoty použijeme `Console.WriteLine(prijmy[0])`.

Poslední část kódu by měla být poměrně jasná, ale pro jistotu si projdeme ještě tu. Ukládáme zde do proměnné prumer typu `double` (jelikož průměrný příjem by mohl vyjít jako desetinné číslo) součet všech prvků z pole vydělený počtem prvků. To je obecně známý vzorec pro výpočet průměru. Zajímavá část tohoto úseku je `prijmy.Length`. `Length` je vlastnost, kterou obsahuje každé pole a její návratovou hodnotou je počet prvků konkrétního pole.

> V jazyce C# mají objekty a třídy své členy přístupné pod tečkou. Člen `Length` je tzv. vlastností pole. Vypisovat všechny tyto vlastnosti by však bylo na velmi dlouho a není to nutné, jelikož ve vývojovém prostředí je k dispozici intellisense, která vám vždy nabídne všechny tyto členy k výběru. Co je to objekt, třída nebo vlastnost nyní ponecháme stranou, protože se jimi budeme podrobněji zabývat v dalších kapitolách.

> V tomto příkladu bychom mohli použít místo vlastnosti `Length` rovnou číslo `5`. V rozsáhlejších aplikacích je však vhodné použít zmíněnou vlastnost, protože pokud bychom chtěli později spočítat průměr například pro 6 osob, ubývá nám místo, kde musíme modifikovat kód a zlepšuje se nám tedy přehlednost a udržitelnost našeho řešení.

Inicializace pole

Zapsáním `int[] array = new int[5]`; se vytvoří pole se jménem `array` o pěti prvcích, jak jsme již zmiňovali výše. Co jsme však nezmínili je, že všechny tyto prvky jsou implicitně nastaveny na určitou defaultní hodnotu. Například pro `int` je to 0, pro `string` je to prázdný řetězec (`""`). Pole lze však také inicializovat přímo jeho hodnotami. Pojďme se podívat, jak by to vypadalo pro náš předchozí příklad:

Příklad

```
// Inicializace pole s přednastavenými hodnotami
int[] prijmy = new int[] { 20000, 15000, 25000, 30000, 12000 };


// Výpočet průměru podle vzorce
double prumer = (prijmy[0] + prijmy[1] + prijmy[2] + prijmy[3] + prijmy[4]) / prijmy.Length;
```

```
// Výpis průměru
Console.WriteLine(prumer);
```

Tento zápis je o poznání kratší, avšak někomu by mohl přijít méně přehledný. Výsledek je však stejný, jako v první příkladu a je tedy na čtenáři, který způsob zvolí.

> Povšimněte si, že v druhém příkladu již ze závorek zmizel počet prvků. Kompilátor si již sám zjistí, pro kolik prvků má alokovat místo podle počtu čísel ve složených závorkách.

List

V této kapitole bychom se však měli také dozvědět, co je to List. Představte si jej, jako takové chytřejší pole. Hlavním rozdílem mezi těmito strukturami je to, že pro pole je nutné znát počet prvků již při inicializaci. Pro list ovšem nikoli. Další zásadní rozdíl je, že list je narozdíl od pole `generická datová struktura`. Co to přesně znamená teď není důležité. Nyní je pro nás důležité si pamatovat, že u generických datových struktur píšeme vždy za název struktury špičaté závorky (`<>`) a do nich datový typ prvků uložených v takové struktuře. Dalšími rozdíly jsou odlišné přidávání a mazání prvků. Do pole není možné přidávat další prvky, ani žádné mazat. Jediný způsob, jak nějaký prvek smazat je vynulovat jej. To však nemusí být vždy ideální. Pojďme se teď tedy ještě jednou vrátit k našemu příkladu a přepsat jej s použitím datové struktury `List<>`:

Příklad

```
// Založení nového listu. Důležitý je datový typ int ve
// špičatých závorkách kterým říkáme, že náš list bude
// obsahovat pouze číselné hodnoty. Kombinace více
// datových typů není v jazyce C# možná.
List<int> prijmy = new List<int>();

// Přidávání a odebírání hodnot listu. V případě,
// že list obsahuje více shodných hodnot, které se snažíme
// odebrat, je vždy odstraněn prvek s větším indexem (naposledy
// přidaný shodný prvek)
prijmy.Add(20000);
prijmy.Add(15000);
prijmy.Add(25000);
prijmy.Add(30000);
prijmy.Add(12000);
prijmy.Add(40000);
prijmy.Remove(40000);
```

```
prijmy.Add(40000);
prijmy.RemoveAt(5);

// Výpočet průměru. K hodnotám v listu
// můžeme přistupovat jako k prvkům v poli
// za pomocí hranatých závorek
double prumer = (prijmy[0] + prijmy[1] + prijmy[2] + prijmy[3] + prijmy[4])/prijmy.Count;

// Výpis výsledku
Console.WriteLine(prumer);
```

> Pro práci s listy je nutné nainportovat si knihovnu `System.Collections.Generic`! Knihovna může být naimportována ručně vložením using-u nad namespace nebo automaticky za použití vývojového prostředí (např. Visual Studio), které by vám mělo samo automaticky using nabídnout.

Import knihovny

```
using System;
using System.Collections.Generic; // Důležité!
```

Tento kód nám vrátí opět stejný výsledek `20 400`. Pro demonstraci mazání jsme zde dvakrát přidali další hodnotu, kterou jsme následně smazali. Jak si můžete povšimnout, pro přidávání slouží metoda `Add()`. Pro mazání `Remove()`, nebo `RemoveAt()`. Metoda `Remove()` se chová jinak, než při přístupu k prvku v poli přes index. Zde místo indexu používáme přímo hodnotu, kterou chceme odstranit. `RemoveAt()` se na druhé straně chová stejně, jako bychom k prvku přistupovali v poli přes index. Prvky v Listu jsou opět indexovány od `0` do `n-1`, jako tomu bylo v poli. Pokud však odstraníme prvek například na indexu 2, všechny další prvky se přesunou. Pokud tedy máme prvky s indexy 0, 1, 2, 3, 4 a odstraníme prvek na indexu 2, budeme mít indexy 0, 1, 2, 3. Přístup k datům a jejich změna je stejná jako u pole. Tedy za použití hranatých závorek (`[]`) a indexu prvku.

Na závěr si povšimněte, že při inicializaci Listu zapisujeme `new List<int>();`! Kulaté závorky na konci jsou nezbytné. Narozdíl od pole se zde totiž volá konstruktor třídy `List`. Co je to konstruktor si vysvětlíme až v druhém ročníku. Nyní nám stačí vědět, že při inicializaci proměnných některých datových typů je potřeba napsat nakonec kulaté závorky (`()`), a že `List` je jedním z nich.

> Všimněte si také, že zde není nutné znát předem počet osob v naší skupině. Pokud bychom tento počet tedy neznali, použili bychom právě list.

> Všimněte si také, že v této ukázce se nám metoda `prijmy.Length` změnila na `prijmy.Count`. Je to proto, že List nemá vlastnost `Length`, ale má vlastnost `Count`, která vrací stejnou

hodnotu jako `Length` pro pole - celkový počet prvků v listu.

Referenční a primitivní datové typy

V předchozích podkapitolách jsme si představili pole a list a ukázali jsme si, jak s nimi můžeme pracovat. V této kapitole bych rád poukázal na častý problém začátečníků, jež plyne z neznalosti problematiky referenčních a primitivních datových typů. Veškeré datové typy, které jsme si do teď ukazovali (např. `int` nebo `char`) byli tzv. primitivní. Oba naše nové datové typy (list a pole) jsou zástupci tzv. referenčních datových typů. Rozdíl mezi těmito dvěma skupinami si budeme demonstrovat na malé ukázce:

Primitivní a referenční typy

```
// Pomocné pole
int[] suda = {2, 4, 6};
int[] licha = {1, 3, 7};

// Přemažeme lichá čísla těmi sudými (nelogičnost tohoto kroku nyní ponechme stranou)
licha = sude;

// Smažeme všechna sudá čísla
Array.Clear(sude, 0, suda.Length);

// Vypíšeme první prvek lichých čísel
Console.WriteLine(licha[0]);
```

Co by bylo výstupem takovéhoto programu? Na posledním řádku vypisujeme první prvek pole `licha`, jež jsme na řádku 6 přepsali polem sudých čísel. Nabízí se tedy odpověď, že výstupem tohoto programu bude číslo `2`, jelikož dvojka v době vykonávání řádku 6 byla prvním prvkem pole `suda`, než později došlo k jeho úplnému promazání. Kdybychom se však podívali na obsah pole licha, zjistíme, že je úplně prázdné. Ačkoliv jsme tedy na řádku 9 pročistili pouze pole `suda`, nějakým způsobem došlo i k promazání pole `licha`. Proč?

Odpověď na tuto otázku bychom nalezli na řádku 6, kde ve skutečnosti nepřepisujeme pole licha (1, 3, 7) polem sude (2, 4, 6) nýbrž tímto zápisem říkáme, že proměnná `licha` bude nyní odkazovat na proměnnou `suda`. Došlo tedy k předání tzv. `reference` (odkazu) na místo `hodnoty`, jak jsme zvyklí u primitivních datových typů. To je také důvod, proč někdy těmto typům říkáme odkazové a hodnotové.

Problematika primitivních a referenčních datových typů může být pro začátečníka velmi zavádějící a budeme se jí více věnovat v poslední kapitole.

> Pokud bychom chtěli doopravdy nahradit obsah pole licha obsahem pole sude (na místo pouhého předání odkazu / reference), můžeme k tomuto účelu využít metodu `CopyTo` - `sude.CopyTo(licha , 0)`;

Shrnutí

V této kapitole jsme si ukázali list a pole. Tyto konstrukty jsou nejužitečnější v kombinaci s tzv. cykly, které si ukážeme v příští kapitole. Níže jsou pak pro zopakování užitečné příkazy pro práci s poli a listy:

Užitečné ukázky

```
// Základní inicializace pole o 10 prvcích:
int[] pole = new int[9];

// Inicializace pole i s jeho hodnotami:
int[] pole = { 3, 7, 69 };

// Získání hodnoty z pole:
int prvek = pole[0];

// Nastavení hodnoty na libovolném indexu:
pole[0] = 32;

// Inicializace listu:
List<int> list = new List<int>();

// Přístup k prvkům listu probíhá stejně jako u pole:
int prvek = list[0];

// Přidání hodnoty do listu:
list.Add(32)

// Odebrání hodnoty z listu:
list.Remove(11);
```

Revision #1
Created 2025-05-21 08:12:41 UTC by Magdalena Dobešová
Updated 2025-05-21 08:29:14 UTC by Magdalena Dobešová