**Základy programování a algoritmizace**

**Kolekce**

**Erik Král**



**2020**

**Informace o autorech:**

Ing. et Ing. Erik Král, Ph.D.

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Nad Stráněmi 4511

760 05 Zlín

ekral@utb.cz

OBSAH

[OBSAH 3](#_Toc60863442)

[1 Úvod 4](#_Toc60863443)

[1.1 Jednorozměrná pole 4](#_Toc60863444)

[1.2 Multidimensional array a Jagged array 7](#_Toc60863445)

[1.3 Kolekce 9](#_Toc60863446)

[1.4 Řešené příklady 12](#_Toc60863447)

[Seznam použité literatury 20](#_Toc60863448)

# Úvod

V tomto materiálu probereme jednorozměrná pole s pevnou délkou [1], multidimensional [2] a jagged arrays [3] a kolekce [4] dynamické pole *List* [5] a asociativní pole *Dictionary* [6].

## Jednorozměrná pole

Nejprve probereme **jednorozměrné pole** s pevnou délkou. Typ pole vytvoříme tak, že za typ prvků v poli přidáme znaky *[]*, například pole celých čísel *int* zapíšeme jako *int[]*. Následující příkaz definuje proměnnou typu pole *int[]* a protože pole je referenční typ, tak mu můžeme přiřadit hodnotu *null* což znamená, že nemá ještě přiřazené žádné hodnoty.

int[] pole = null;

Paměť pro vlastní hodnoty prvků potom alokujeme pomocí příkazu *new int[3]* kdy hodnota 3 je počet prvků. Proměnná *pole* potom představuje referenci na zásobníku na data alokované na haldě.

pole = new int[3];

Oba příkazy můžeme sloučit do jednoho zápisu:

int[] pole = new int[3];

a hodnoty můžeme inicializovat pole pomocí zápisu *{ 1, 2, 3 }*.

int[] pole = new int[3] { 1, 2, 3 };

Předchozí zápis můžeme různým způsobem zkrátit. Všechny následující zápisy mají stejný výsledek:

int[] pole1 = new int[3] { 1, 2, 3 };

int[] pole2 = new int[] { 1, 2, 3 };

int[] pole3 = new [] { 1, 2, 3 };

int[] pole4 = { 1, 2, 3 };

K jednotlivým prvků poli přistupujeme pomocí hranatých závorek [], kdy index začíná od nuly:

int[] pole = new int[3] { 1, 2, 3 };

Console.WriteLine(pole**[0]**); // prvni prvek

Console.WriteLine(pole**[1]**); // druhy prvek

Console.WriteLine(pole**[2]**); // treti prvek

Delku pole zjistíme pomocí property *pole.Length*. Následující příkaz nastaví hodnoty pole na *0,1,2*:

int[] pole = new int[3];

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

{

    pole[i] = i;

}

Pokud přiřadíme hodnotu pole jinému poli tak předáme referenci na stejné prvky v poli. V následujícím příkazu tedy *pole1* i *pole2* mají referenci na stejná data:

int[] pole1 = new int[3] { 1, 2, 3 };

int[] pole2 = pole1;

pole2[1] = 0;

Console.WriteLine(string.Join(",", pole1));

Console.WriteLine(string.Join(",", pole2));

Pokud chceme vytvořit nezávislé kopie, tak máme několik možností. Nejprve si nadefinujme originální pole:

int[] original = new int[3] { 1, 2, 3 };

První možností je použití **metody** ***ToArray*** z knihovny *LINQ* a vyžadují použití using *System.Linq*.

int[] kopie = original.ToArray();

Další možností je použití **členské metody *CopyTo*** třídy ***Array***. Nejprve si nadefinujeme nové pole se stejným počtem prvků jako má orginál a potom pomocí metody *CopyTo* zkopírujeme prvky z originálního pole do kopie.

int[] kopie = new int[original.Length];

original.**CopyTo**(kopie, 0);

Třetí možností je použití **statické metody *Copy*** třídy ***Array***. Opět si nadefinujeme nové pole se stejným počtem prvků jako má orginál a potom pomocí metody ***Copy*** zkopírujeme prvky z originálního pole do kopie.

int[] kopie = new int[original.Length];

Array.**Copy**(original, kopie, original.Length);

Poslední, co si ukážeme u jednorozměrných polí je, že můžeme vypsat na konzoli s využitím příkazu *string.Join*. Tento výpis je užitečný především při ladění programu, znak "," představuje oddělovač vypsaných prvků na řádku konzole.

Console.WriteLine(string.Join(",", pole));

## Multidimensional array a Jagged array

U vice rozměrných polí máme dvě možnosti Multidimensional array [2] a Jagged array [3].

**Multidimensional array** se definuje následujícím způsobem kdy v tomto příkladu říkáme, že chceme mít dvě pole o třech prvcích:

int[,] multidimensionalArray = new int[2, 3]

{

    { 1, 2, 3 },

    { 4, 5, 6 }

};

Rozměry multidimensional array získáme pomocí metody GetLength, kdy argumentem je index dimenze. V následujícím případě **GetLength(0)** vrátí hodnotu *2* a **GetLength(1)** vrátí hodnotu *3*.

for (int i = 0; i < multidimensionalArray.**GetLength(0)**; i++)

{

    for (int j = 0; j < multidimensionalArray.**GetLength(1)**; j++)

    {

        int prvek = multidimensionalArray[i, j];

        Console.Write($"{prvek} ");

    }

    Console.WriteLine();

}

Property *Length* by nám vrátila celkový počet prvků, tedy 2 x 3 = 6 prvků.

int celkovyPocet = multidimensionalArray.Length;

Oproti tomu **Jagged array** představuje pole referencí na další pole, která pak musíme inicializovat zvlášť. Každý řádek potom může mít různý počet prvků.

Nejprve si nadefinujeme pole referencí:

int[][] jaggedArray = new int[2][];

A potom alokujeme paměť pro jednotlivé řádky, tedy jednorozměrné pole:

jaggedArray[0] = new int[] { 1, 2, 3 };

jaggedArray[1] = new int[] { 4, 5 };

Potom procházíme jednotlivé řádky každý řádek procházíme jako jednorozměrné pole:

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

{

**int[] radek = jaggedArray[i];**

    for (int j = 0; j < radek.Length; j++)

    {

        int prvek = radek[j];

        Console.Write($"{prvek} ");

    }

    Console.WriteLine();

}

Jagged array je rychlejší pro sekvenční přístup k prvkům než Multidimensional array, protože prakticky po získání refence na řádek můžeme procházet jednorozměrné pole prvek po prvku.

## Kolekce

Poslední dva typy, které probereme je dynamické pole *List* a asociativní pole *Dictionary*.

Generická **třída List<int>** představuje dynamické pole, tedy pole do kterého můžeme přidávat nové prvky a také prvky odebírat. Jde o nejčastěji používaný typ kolekcí.

Instanci třídy *List<T>* nadefinujeme následujícím způsobem:

List<int> listCisel = new List<int> { 1, 2, 3, 4, 5, 6 };

Počet prvků zjistíme, na rozdíl od pole, pomocí property Count.

int pocetPrvku = listCisel.Count;

Pro přístup k prvkům můžeme opět použít operátor indexace *[]*.

for (int i = 0; i < pocetPrvku; i++)

{

    int prvek = listCisel**[i]**;

    Console.WriteLine(prvek);

}

Pokud chceme projít všechny prvky, tak nejčastějši používáme příkaz *foreach*, který projde všechny prvky.

foreach (int prvek in listCisel)

{

    Console.WriteLine(prvek);

}

Do dynamického pole můžeme prvky přidat jak na konec tak na libovolný index.

listCisel.Add(4); // pridam na konec

listCisel.Insert(0, 6); // vlozim na zacatek

listCisel.Insert(2, 7); // vlozim na index 2

Také můžeme odstranit prvek s určitou hodnotou, nebo odebrat prvek z určitého idnexu.

listCisel.Remove(7);    // odstrani prvek s hodnotou 7

listCisel.RemoveAt(0);  // odstrani prvek s indexem 0

Také můžeme odstranit všechny prvky v poli pomocí metody Clear.

listCisel.Clear();

Generická třída *Dictionary<TKey, TValue>* představuje kolekcí klíčů a hodnot. Například můžeme mít kolekci studentů, kde klíčem bude identifikační číslo studenta a hodnotou jméno studenta.

Dictionary<int, string> studenti = new Dictionary<int, string>()

{

    [10] = "Petr",

    [20] = "Alena"

};

Nové studenty potom můžeme přidávat s pomocí metody *Add*.

studenti.Add(100, "Jan");

studenti.Add(128, "Jiri");

Pokud by student s daným záznamem už v kolekci existoval, tak program vyvolá výjimku. Proto je vhodnější předem ověřit, zda daný klíč už existuje. A to buď starším způsobem:

if (!studenti.ContainsKey(128))

{

    studenti.Add(128, "Jana");

}

Nebo novějším způsobem:

bool okAdd = studenti.TryAdd(128, "Jana");

Z kolekce můžeme odstranit záznam dle konrétního klíče, například:

studenti.Remove(128);

Pokud bychom chtěli procházet všechny záznamy v *Dictionary<TKey, TValue>*, tak musíme využít typ *KeyValuePair< TKey, TValue>* tedy pár klíč a hodnota, například:

foreach (KeyValuePair<int, string> par in studenti)

{

    Console.WriteLine($"{par.Key}: {par.Value}");

}

## Řešené příklady

Nyní si ukážeme příklady s kompletním kódem a případně okomentujeme jednotlivá řešení. První příklad demonstruje **definici a inicializac**i pole.

using System;

namespace Priprava

{

    class Program

    {

        static void Main(string[] args)

        {

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

            Console.WriteLine(pole[0]);

            Console.WriteLine(pole[1]);

            Console.WriteLine(pole[2]);

            int[] pole1 = new int[3];

            int[] pole2 = new int[] { 1, 2, 3 };

            int[] pole3 = new[] { 1, 2, 3 };

            int[] pole4 = { 1, 2, 3 };

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

            {

                int prvek = pole[i];

                Console.WriteLine($"prvek s indexem {i} ma hodnotu {prvek}");

            }

            foreach (int prvek in pole)

            {

                Console.WriteLine(prvek);

            }

        }

    }

}

Prvky bez inicializace budou mít výchozí hodnotu *0*:

int[] pole1 = new int[3];

Druhý příklad demonstruje možnosti vytvoření **nezávislé (hluboké) kopie pole**.

using System;

using System.Linq;

namespace Test

{

    class Program

    {

        static void Main(string[] args)

        {

            int[] original = new int[3] { 1, 2, 3 };

            int[] kopie1 = original.ToArray();

            int[] kopie2 = new int[original.Length];

            original.CopyTo(kopie2, 0);

            int[] kopie3 = new int[original.Length];

            Array.Copy(original, kopie3, original.Length);

            original[1] = 0;

            Console.WriteLine(string.Join(",", original));

            Console.WriteLine(string.Join(",", kopie1));

            Console.WriteLine(string.Join(",", kopie2));

            Console.WriteLine(string.Join(",", kopie3));

        }

    }

}

Třetí příklad spočítá sumu prvků v poli s pomocí proměnné *suma* a cyklu *for*.

using System;

namespace Priprava

{

    class Program

    {

        static void Main(string[] args)

        {

            int[] pole = { 5, 7, 1, 2, 3 };

            int suma = 0;

            foreach (int prvek in pole)

            {

                Console.WriteLine(prvek);

                suma += prvek;

            }

            Console.WriteLine($"soucet prvku v poli je {suma}");

        }

    }

}

Alternativně můžeme sumu spočítat pomocí metody Sum z knihovny LINQ.

using System;

using System.Linq;

namespace Priprava

{

    class Program

    {

        static void Main(string[] args)

        {

            int[] pole = { 5, 7, 1, 2, 3 };

            int suma = pole.Sum();

            Console.WriteLine($"soucet prvku v poli je {suma}");

        }

    }

}

Další příklad změní a vypíše prvky v poli v obráceném pořadí.

using System;

namespace Priprava

{

    class Program

    {

        static void Main(string[] args)

        {

            int[] pole = { 1, 2, 7, 5, 6, 3, 8 };

            Console.WriteLine(string.Join(",", pole));

            int prvniIndex = 0;

            int posledniIndex = pole.Length - 1;

            for (int i = 0; i < pole.Length / 2; i++)

            {

                int tmp = pole[posledniIndex];

                pole[posledniIndex] = pole[prvniIndex];

                pole[prvniIndex] = tmp;

                ++prvniIndex;

                --posledniIndex;

            }

            Console.WriteLine(string.Join(",", pole));

        }

    }

}

Následující příklad je **bonusový** a představuje variantu předcházející příklad, ale tentokrát v cyklu *for* definuje oba indexy.

using System;

namespace Priprava

{

    class Program

    {

        static void Main(string[] args)

        {

            int[] pole = { 1, 2, 7, 5, 6, 3, 8 };

            Console.WriteLine(string.Join(",", pole));

            for (int i = 0, j = pole.Length - 1; i < pole.Length / 2; i++, j--)

            {

                int tmp = pole[j];

                pole[j] = pole[i];

                pole[i] = tmp;

            }

            Console.WriteLine(string.Join(",", pole));

        }

    }

}

Konrétně si prostudujte následující syntaxi cyklu *for*:

for (int i = 0, j = pole.Length - 1; i < pole.Length / 2; i++, j--)

Následující příklad demostruje přáci s generickou třídou *List<T>*, kdy definujeme pole studentů. Typ Student je třída a třídy probereme později.

using System;

using System.Collections.Generic;

namespace Property

{

    class Student

    {

        public string Jmeno { get; set; }

        public Student(string jmeno)

        {

            Jmeno = jmeno;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            List<Student> studenti = new List<Student>

            {

                new Student("Jana"),

                new Student("Bozena"),

                new Student("Xenie")

            };

            Student jirka = new Student("Jirka");

            studenti.Add(jirka);

            foreach (Student student in studenti)

            {

                Console.WriteLine(student.Jmeno);

            }

            Console.WriteLine("Odstraneni studenta se jmenem Jirka");

            studenti.Remove(jirka); // odstranim dle reference

            foreach (Student student in studenti)

            {

                Console.WriteLine(student.Jmeno);

            }

        }

    }

}

Seznam použité literatury

1. Jednorozměrná pole – Průvodce programováním v C# | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 05.01.2021]. Dostupné z: <https://docs.microsoft.com/cs-cz/dotnet/csharp/programming-guide/arrays/single-dimensional-arrays>.
2. Multidimensional Arrays - C# Programming Guide | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 05.01.2021]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/multidimensional-arrays>
3. Jagged Arrays - C# Programming Guide | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 05.01.2021]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/jagged-arrays>
4. Collections (C#) | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 05.01.2021]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/collections>
5. List Class (System.Collections.Generic) | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 05.01.2021]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-5.0>
6. Dictionary Class (System.Collections.Generic) | Microsoft Docs. [online]. Copyright © Microsoft 2021 [cit. 05.01.2021]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=net-5.0