ŘÍZENÍ BĚHU PROGRAMU

pole

příkazy větvení

příkazy cyklu

Pole

- **pole** je datová struktura, která obsahuje více položek stejného datového typu, položky jsou v paměti řazeny za sebou
- k jednotlivým položkám se přistupuje pomocí **indexu**, což je celočíselná proměnná
- **v případě, že má pole *n* prvků, první prvek pole má index 0, poslední pak *n*-1**
- Deklarace pole:

datovýTyp[] *názevPole*;

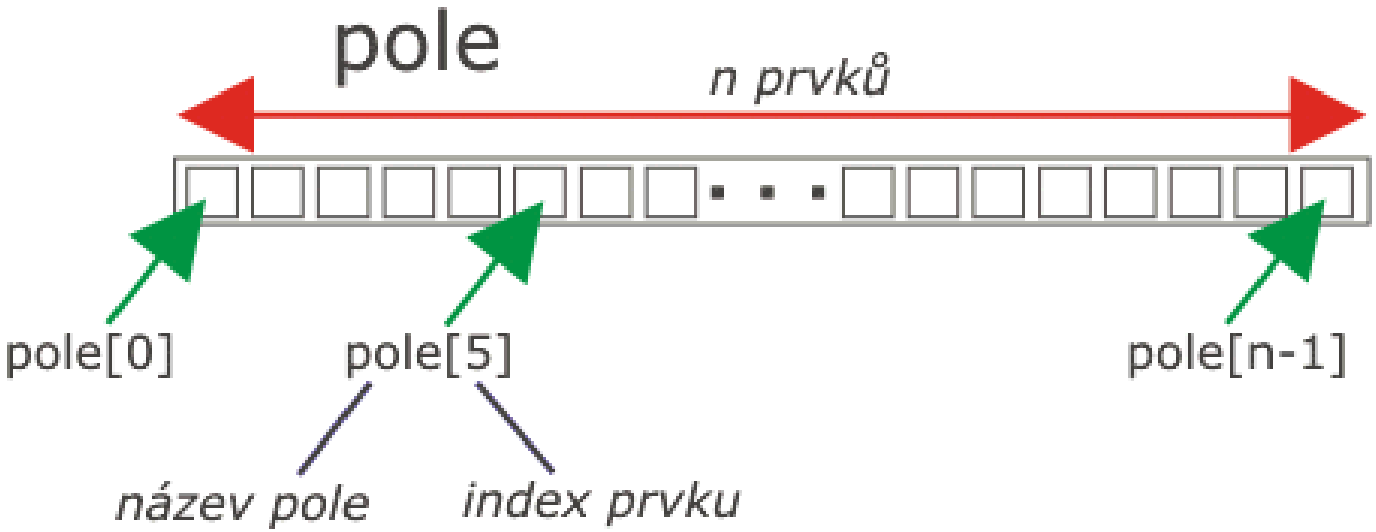
- Přidělení paměti pro pole:

datovýTyp[] *názevPole* = new *datovýTyp*[*početPrvků*];

- prvek pole s indexem i

názevPole[*i*];

Pole



Pole

- Uvažujme program, ve kterém jsou uloženy ceny jednotlivých výrobků v poli. Pokud ceny uložíme jako typ double a výrobků je 10, bude deklarace pole vypadat následovně:

```
double[] ceny = new double[10];
```

- prvek s indexem 4, tj. cenu 5. výrobku (!), vypíšeme následovně:

```
Console.WriteLine(ceny[4]);
```

- celková cena všech výrobků je:

```
double celkovaCena = ceny[0] + ceny[1] +
  ceny[2] + ceny[3] + ceny[4] + ceny[5] +
  ceny[6] + ceny[7] + ceny[8] + ceny[9];
```

Pole

- pole v C# je odvozené od třídy **System.Array**
- třída System.Array obsahuje užitečné metody a vlastnosti pro práci s polem:

```
Array.Copy(prvniPole, druhePole);
Array.Clear(mojePole);
mojePole.Length;
```

- vícerozměrná pole (např. pro práci s maticemi, zejména v grafických aplikacích)

Deklarace:

datovýTyp[,] názevPole = new datovýTyp[početŘádků, početSloupců];

- přístup k položkám je potom obdobný jako u jednorozměrných polí

Řízení běhu programu

1. **příkazy větvení**
   - velmi často je potřeba tok programu rozvětvit na dvě nebo více větví (např. zadáte správné heslo a program se ubírá jednou větví a nebo zadáte špatné heslo a program reaguje jinou větví)
2. **příkazy cyklu**
   - v mnoha případech se některé příkazy provádějí opakovaně
   - například vyhledávání údaje v nějaké množině dat může probíhat tak, že je každý záznam porovnáván s vyhledávaným údajem
   - jedná se o opakování stejného příkazu nebo několika příkazů

Příkazy větvení – příkaz **if**

- požadujeme určitý blok příkazů provést pouze v případě, že je pravdivá určitá podmínka

Syntaxe:

```
if (podmínka)
  {
     blok příkazů
  }
```

- **podmínku** musí tvořit **logický výraz**, tj. takový, který se vyhodnotí jako **true** nebo **false**

Příkazy větvení – příkaz **if**

```
if (zadaneCislo == 0)
      {
       // Tento blok kódu se provede,
       // pokud je podmínka vyhodnocena jako true.
         Console.WriteLine("Pozor! Nulova hodnota");
       }
```

- všimněte si operátoru == v podmínce, jedná se o porovnávací operátor, vyhodnotí se jako **true**, pokud se obě strany rovnají, v opačném případě se vyhodnotí jako **false**
- nezaměňujte operátor porovnání s operátorem přiřazení, který je tvořen pouze jedním rovnítkem
- jednoduchý příkaz **if** lze psát do jednoho řádku

Příkaz větvení – příkaz **if** / **else**

- program se podle podmínky rozdělí na dvě větve, přičemž první se provede, pokud je podmínka pravdivá, druhá větev se provede, pokud je podmínka nepravdivá

Syntaxe:

```
if (podmínka)
   {
      blok příkazů, které se mají provést
      v případě vyhodnocení podmínky jako true
   }
else
   {
      blok příkazů, které se mají provést
      v případě vyhodnocení podmínky jako false
   }
```

Příkaz větvení – příkaz **if / else if / else**

- v případě dělení do více větví dle podmínek

Syntaxe:

```
if (podmínka1){
      blok příkazů, který se provede  v případě vyhodnocení podmínky1 jako
    true
    }
else if (podmínka2){
      blok příkazů, který se provede v případě vyhodnocení podmínky2 jako
    true
    }
    ...
else if (podmínkaN){
      blok příkazů, který se provede v případě vyhodnocení podmínkyN jako
    true
    }
else{
     blok příkazů, který se provede v případě vyhodnocení všech podmínek
    jako false
    }
```

Příkaz větvení – příkaz **switch**

- potřebujeme program rozvětvit podle hodnoty určité proměnné

Syntaxe:

```
switch (testovaný výraz){
    case hodnota1:
        //  blok příkazů se provede, jestliže výraz nabývá hodnotu hodnota1.
         break;
  case hodnota2:
        //  blok příkazů se provede, jestliže výraz nabývá hodnotu hodnota2.
         break;
        // ...
  default:
         //  blok příkazů se provede,
         // jestliže testovaný výraz nenabývá žádnou z předchozích hodnot.
         break;
   }
```

Příkazy cyklu

- příkaz **for**
  - blok příkazů provádí opakovaně v závislosti na hodnotě proměnné zvané čítač
- příkaz **while**
  - provádí příkazy opakovaně, přičemž podmínka pro další opakování je umístěna na začátku smyčky
- příkaz **do while**
  - provádí blok příkazů opakovaně, přičemž řídící podmínka pro další opakování je umístěna na konci smyčky.
- příkaz **foreach**
  - pro každý prvek pole nebo kolekce, ale není možné měnit prvky kolekce

Příkazy **continue** a **break**

- další:
  - příkaz **break**
    - příkaz, který umožní násilné ukončení opakování bloku příkazů v cyklu
    - cyklus se ukončí a program pokračuje prvním příkazem za cyklem
  - příkaz **continue**
    - příkazem, pomocí něhož můžete násilně ukončit aktuální iteraci cyklu a pokračovat následující

Příkaz cyklu – příkaz **for**

- **Syntaxe:**

```
for ( inicializace čítače ; podmínka ; aktualizace )
      {
      // příkazy (tělo cyklu)
      }
```

- jednoduchý cyklus můžete psát do jednoho řádku

```
for(int i=1;i<=14;i++)Console.WriteLine("{0}.týden", i);
```

Příkaz cyklu – příkaz **for**

- příkaz začne tzv. **inicializací** (**inicializací čítače**) - **čítač** (**counter**) je řídící proměnná cyklu - pro každou iteraci cyklu má určitou hodnotu a většinou řídí, kdy se cyklus ukončí

vlastní cyklus:

1. **vyhodnocení podmínky (testovací část)** - podmínka většinou obsahuje hodnotu čítače; pokud se vyhodnotí jako **true**, následuje provedení těla cyklu, v opačném případě se příkaz ukončí a program pokračuje prvním příkazem za příkazem `for`
2. **provedení těla cyklu** - tělo cyklu je blok příkazů; provedou se pouze tehdy, pokud je podmínka vyhodnocena jako **true**
3. **aktualizace (zpravidla aktualizace čítače)** - hodnotu čítače nebo jiného identifikátoru stavu, na kterém je závislé vyhodnocení podmínky, je potřeba změnit, neboť v opačném případě by se podmínka stále vyhodnocovala stejně - v případě hodnoty true bychom dostali nekonečný cyklus

Příkaz cyklu - **while**

- cyklus **while** závisí pouze na podmínce, která se vyhodnotí vždy na začátku iterace

Syntaxe:

```
while (podmínka)
    {
      tělo cyklu
    }
```

- je důležité si uvědomit, že někde uvnitř programu **musí dojít ke změně hodnot**, které se vyskytují ve výrazu podmínky
- podmínka by se měla někdy vyhodnotit jako **false**, aby se příkaz ukončil
- jinak vytvoříte nekonečný cyklus

Příkaz cyklu – **do while**

- **do while** je podobný cyklu while, podmínka se však v tomto případě vyhodnotí vždy na konci iterace

Syntaxe:

```
do
    {
        tělo cyklu
    } while (podmínka);
```

- v cyklu **do while** se provede smyčka **alespoň jednou**, jelikož podmínka se testuje až na konci

Příkaz cyklu - **foreach**

- pro každý prvek pole nebo kolekce

```
int[] poleFi = new int[]{ 0, 1, 2, 3, 5, 8, 13 };
            foreach (int polozka in poleFi)
             {
                 System.Console.WriteLine(polozka);
             }
```

- můžete zjistit informace, vlastnosti...
- nemůžete měnit prvky pole nebo prvky kolekce

Příkaz **break**

- jakmile se v těle cyklu vyhodnotí příkaz **break**, **cyklus se ukončuje a pokračuje se následujícím příkazem za cyklem**
- častým případem užití příkazu break, je situace, kdy procházíme v cyklu hodnoty a zpracováváme je, dokud se nevyskytne určitá hodnota - například ta, kterou hledáme
- nebo, když výpočet je rychlý a dosáhneme splnění nějaké důležité podmínky, např. velikosti chyby

Příkaz **continue**

- pomocí příkazu **continue** lze **přeskočit zbytek** provádění těla aktuální iterace cyklu a pokračovat následující iterací
- příklad

```
string txt = Console.ReadLine();
for (int i = 0; i < txt.Length; i++)
 {
 if (txt[i] == '@')continue;
 Console.WriteLine("{0}:{1}„,txt[i],(int)txt[i]);
 }
```

Příští týden: Výjimky