KAPITOLA 4

Cykly a pole

V této kapitole:

- Cyklus while
- Cyklus do-while
- Cyklus for
- Pole
- Kontrolní otázky
- Řešená cvičení
- Cvičení

Velmi často se setkáte s tím, že budete potřebovat, aby program určité příkazy vykonal opakovaně. Představte si situaci, kdy budete chtít vytvořit program, který uživateli umožní sečíst libovolné množství čísel. S obdobnou úlohou byste si již poradili za předpokladu, že by bylo pevně dáno, kolik čísel budete sčítat. Pokud počet čísel neznáte a chcete, aby program fungoval tak, že se po každém zadaném čísle zeptá, zdali chce uživatel zadat další, musíte se naučit další konstrukce jazyka C# umožňující určitou sadu příkazů vykonat opakovaně. Tyto konstrukce nazýváme cykly. Zároveň se naučíte používat nový datový typ pole, který vám umožní si do proměnné uložit v podstatě libovolné množství dat.

Cyklus while

Cyklus `while` funguje tak, že definujete výraz, který se před každým průběhem cyklu vyhodnotí, a pokud je vyhodnocen jako pravdivý, příkazy uvedené v těle cyklu se vykonají. Průběhem cyklu tedy rozumíme právě jedno vykonání příkazů v těle cyklu. Cyklus `while` má následující syntaxi:

```
while (logický výraz)
{
  // Příkazy v těle cyklu
}
```

Po skončení jednoho průběhu cyklu se opět vyhodnotí logický výraz. Pokud je vyhodnocen jako pravdivý, příkazy v těle cyklu se vykonají znovu. Snadno nahlédneme, že pokud by byl výraz vyhodnocen vždy jako pravdivý, bude se cyklus vykonávat donekonečna. Na toto je potřeba dávat pozor, protože nekonečný cyklus způsobí, že program nikdy neskončí. Pojďme si cyklus `while` vyzkoušet na následujícím příkladu. Vezměme příklad z podkapitoly Příkaz switch a upravme jej tak, že program umožní provádět operace opakovaně, aniž by si pamatoval

mezivýsledek. Po každém provedeném výpočtu se program uživatele zeptá, zdali chce provést další výpočet.

```
string provestVypocet = "ano";
while (provestVypocet == "ano")
{
  Console.Write("Zadejte prvni cislo: ");
  int x = Int32.Parse(Console.ReadLine());
  Console.Write("Zadejte druhe cislo: ");
  int y = Int32.Parse(Console.ReadLine());
  Console.Write("Zadejte pozadovanou operaci ( + - * / ): ");
  string operace = Console.ReadLine();

  switch (operace)
  {
    case "+":
      Console.WriteLine("{0} + {1} = {2}", x, y, x + y);
      break;
    case "-":
      Console.WriteLine("{0} - {1} = {2}", x, y, x - y);
      break;
    case "*":
      Console.WriteLine("{0} * {1} = {2}", x, y, x * y);
      break;
    case "/":
      Console.WriteLine("{0} / {1} = {2}", x, y, x / y);
      break;
    default:
      Console.WriteLine("Byla zadana neznama operace.");
      break;
  }
  Console.Write("Chce provest dalsi vypocet? (ano/ne): ");
  provestVypocet = Console.ReadLine();
}
```

```
Zadejte prvni cislo: 3
Zadejte druhe cislo: 5
Zadejte pozadovanou operaci ( + - * / ): *
3 * 5 = 15
Chce provest dalsi vypocet? (ano/ne): ano
Zadejte prvni cislo: 20
Zadejte druhe cislo: 10
Zadejte pozadovanou operaci ( + - * / ): -
20 - 10 = 10
Chce provest dalsi vypocet? (ano/ne): ne
```

Věnujme pozornost zejména tomu, že ještě před samotným začátkem cyklu definujeme proměnnou `provestVypocet` a nastavíme jí hodnotu tak, aby se cyklus provedl minimálně jednou. To, kolikrát se poté cyklus zopakuje, určí uživatel tím, že na závěr běhu cyklu zadá ano, pokud bude chtít výpočet opakovat.

Podívejme se na následující příklad, na kterém si také procvičíte cyklus `while`. Napišme program, který pro zadané celé číslo spočítá součet všech celých čísel větších než nula a menších než zadané číslo.

```
int soucet = 0;
Console.Write("Zadejte cislo: ");
int cislo = int.Parse(Console.ReadLine());
while (cislo > 0)
{
  Console.WriteLine("Soucet = {0} + {1}", soucet, cislo);
  soucet = soucet + cislo;
  cislo = cislo - 1;
}
Console.WriteLine("Soucet = {0}", soucet);
Console.ReadKey();
```

```
Zadejte cislo: 7
Soucet = 0 + 7
Soucet = 7 + 6
Soucet = 13 + 5
Soucet = 18 + 4
Soucet = 22 + 3
Soucet = 25 + 2
Soucet = 27 + 1
Soucet = 28
```

Věnujte pozornost zejména proměnné `soucet`. Cyklus proběhne tolikrát, kolik bylo uloženo do proměnné `cislo`, protože na konci každého běhu cyklu je hodnota této proměnné snížena o jedničku. V každém běhu cyklu přičteme do proměnné `soucet` aktuálně zpracovávané číslo. Tato proměnná je pro nás pomocným čítačem, bez kterého bychom tuto úlohu nedokázali vyřešit.

Upozornění: Důležitý je fakt, že proměnnou soucet musíme definovat před samotným začátkem cyklu, protože pokud proměnnou definujeme uvnitř těla cyklu, její platnost končí s koncem cyklu samotného. Zamyslete se nad tím, jak tento program napsat tak, abychom obsah proměnné cislo neměnili a nechali si v ní uložené číslo, která zadal uživatel.

Cyklus do-while

Jak již název tohoto cyklu napovídá, má mnoho společného s cyklem `while`. Jediný rozdíl mezi cykly `do-while` a `while` je ten, že u cyklu `do-while` se výraz určující, zdali se cyklus bude opakovat, zapisuje až na konec cyklu. Cyklus `do-while` se tedy vždy vykoná nejméně jednou. Syntaxe tohoto cyklu je následující:

```
do
{
  // Příkazy v těle cyklu
} while (logický výraz)
```

Zkusme si přepsat příklad na provádění výpočtů z podkapitoly Cyklus while pomocí cyklu `do-while`.

```
string provestVypocet;
do
{
  Console.Write("Zadejte prvni cislo: ");
  int x = Int32.Parse(Console.ReadLine());
  Console.Write("Zadejte druhe cislo: ");
  int y = Int32.Parse(Console.ReadLine());
  Console.Write("Zadejte pozadovanou operaci ( + - * / ): ");
  string operace = Console.ReadLine();

  switch (operace)
  {
    case "+":
      Console.WriteLine("{0} + {1} = {2}", x, y, x + y);
      break;
    case "-":
      Console.WriteLine("{0} - {1} = {2}", x, y, x - y);
      break;
    case "*":
      Console.WriteLine("{0} * {1} = {2}", x, y, x * y);
      break;
    case "/":
      Console.WriteLine("{0} / {1} = {2}", x, y, x / y);
      break;
    default:
      Console.WriteLine("Byla zadana neznama operace.");
      break;
  }
  Console.Write("Chce provest dalsi vypocet? (ano/ne): ");
  provestVypocet = Console.ReadLine();
} while (provestVypocet == "ano");
```

Všimněte si, že proměnnou `provestVypocet` definujeme před samotným cyklem – stejně jako u `while` cyklu.

Cyklus for

Tento cyklus vám bude užitečný zejména ve chvíli, kdy budete potřebovat provést předem známý počet opakování cyklu. Cyklus `for` má následující syntaxi.

```
for (inicializační výraz ; ukončující logický výraz ; změna řídící proměnné)
{
  // Příkazy v těle cyklu
}
```

V rámci tohoto cyklu se používá tzv. řídicí proměnná, kterou použijete pro řízení počtu opakování cyklu `for`. Tuto proměnnou nejdříve inicializujete na výchozí hodnotu a po každém průběhu cyklu její hodnotu změníte. Cyklus se opakuje, dokud ukončující logický výraz vrací hodnotu `true`. Podívejte se na následující program, který na příkazový řádek vypíše čísla od nuly do desítky.

```
for (int i = 0; i <= 10; i++)
{
  Console.WriteLine(i);
}
```

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

Nejdříve se vytvoří proměnná `i`, která je typu `Int32`. Cyklus se bude opakovat, dokud je `i` <= 10. Po každém průběhu cyklu se hodnota proměnné `i` zvýší o jedničku. Pojďme nyní napsat program, který nejdříve od uživatele získá počet čísel, která uživatel následně zadá, a na závěr vypíše jejich součet.

```
Console.Write("Zadejte pocet cisel: ");
int pocetCisel = Int32.Parse(Console.ReadLine());
int soucet = 0;
for (int i = 1; i <= pocetCisel; i++)
{
  Console.Write("Zadejte {0}. cislo: ",i);
```

```
  soucet = soucet + Int32.Parse(Console.ReadLine());
}
Console.WriteLine("Soucet zadanych cisel = {0}", soucet);
Zadejte pocet cisel: 5
Zadejte 1. cislo: 5
Zadejte 2. cislo: 10
Zadejte 3. cislo: 3
Zadejte 4. cislo: 7
Zadejte 5. cislo: 20
Soucet zadanych cisel = 45
```

Všimněte si toho, že řídicí proměnnou `i` můžeme inicializovat na libovolné číslo. V našem případě jsme ji inicializovali na jedničku. Počet čísel, která uživatel zadá, jsme použili v ukončovacím logickém výrazu v cyklu `for`. Zároveň se podívejte na elegantní použití řídicí proměnné `i` ve výpisu žádosti o zadání dalšího čísla, kde proměnnou `i` používáme pro číslování toho, kolikáté číslo uživatel zadává. Možná si teď pokládáte otázku, proč máme kromě cyklu `while` i cyklus `for`, když výše uvedené programy je možné zapsat i pomocí cyklu `while`.

Poznámka: Je samozřejmě možné používat pouze jeden typ cyklu, např. cyklus **`while`**. Nicméně z důvodu jednoduššího a přehlednějšího zápisu zdrojového kód je lepší využívat ten typ cyklu, který je v dané situaci vhodnější. Jak bylo řečeno na začátku této kapitoly, cyklus **`for`** je vhodný zejména tehdy, když dopředu znáte počet opakování, která se mají vykonat.

Pole

S dosud nabytými znalostmi jste schopni vytvářet programy, ve kterých víte, kolik dat budete potřebovat uložit do proměnných. Pokud budete mít za úkol napsat program, který od uživatele získá tři čísla, která si bude uchovávat v proměnných, a který vypíše jejich součet, bez větších obtíží to zvládnete. Způsobů, jak takovýto program vytvořit, je více. Řešení by mohlo vypadat např. takto:

```
Console.Write("Zadejte 1. cislo: ");
int cislo1 = Int32.Parse(Console.ReadLine());
Console.Write("Zadejte 2. cislo: ");
int cislo2 = Int32.Parse(Console.ReadLine());
Console.Write("Zadejte 3. cislo: ");
int cislo3 = Int32.Parse(Console.ReadLine());
int soucet = cislo1 + cislo2 + cislo3;
Console.WriteLine("Soucet cisel je {0}", soucet);
Console.ReadKey();
```

Zatím si ale nedokážete poradit s programem, který by se nejdřív uživatele zeptal, kolik čísel má uložit, a následně vypsal jejich součet. Proto se naučíte používat nový datový typ, takzvané pole.