Proměnné, operátory v jazyce C#

Jana Finkeová

Proměnné

- v každém programu je nutné udržovat data
- data ukládáme v proměnných
- **proměnná** představuje určitou část paměti, která je určitého **datového typu** a kterou si vhodně pojmenujeme
- do proměnné můžeme ukládat hodnoty daného datového typu a to opakovaně, **můžeme tedy hodnotu v proměnné měnit**
- **datový typ** určuje jakým způsobem jsou data obsažená v proměnné (tj. **hodnota proměnné**) uložena v paměti a také, jaké operace lze s danou proměnnou provádět
- například proměnné číselného (aritmetického) typu můžeme sčítat, násobit apod., textovou proměnnou násobit nelze

Proměnné

- pro běh programu je významná adresa proměnné v paměti,
- pro programátora je důležitý název proměnné - **identifikátor**

deklarace proměnné:

```
datovýTyp  názevProměnné;
```


Proměnné

```
class Program{
     static void Main()
     {
        int i;
        Console.WriteLine(i);
        Console.ReadLine();
     }
  }
```

Co se stane?

- pokud bychom pracovali s hodnotou proměnné *i* a nebyla v ní uložená hodnota, například bychom zkusili vypsat její hodnotu, nastane při překladu chyba
- (pokud si deklarujeme v programu proměnnou, kterou nepoužijeme, vypíše překladač varování)

Proměnné

- dříve než se proměnná použije, musí mít přiřazenou hodnotu
- jedná se o jedno z mnoha opatření, kterými se jazyk C# snaží zabránit nechtěným chybám
- do již deklarované proměnné můžeme **přiřadit hodnotu** pomocí **přiřazovacího příkazu**

přiřazovací příkaz :

```
názevProměnné = hodnotaProměnné;
jinaPromenna = vyraz;
i = 5;
C = 1.23456;
i = 2*i + 1;
```


Proměnné

```
class Program
     {
        static void Main()
        {
           int i;
           i = 5;
           Console.WriteLine(i);
           j = 15;
           Console.ReadLine();
        }
     }
```

Co se stane?

- pokud bychom se snažili přiřadit hodnotu do proměnné, která není deklarována, nahlásí kompilátor chybu

Proměnné

- pokud již při deklaraci proměnné víte, jaká počáteční hodnota bude v této proměnné, přiřaďte ji hned
- pak hovoříme o **inicializaci proměnné** a odpovídající příkaz se nazývá **inicializační**

inicializační příkaz:

```
datovýTyp názevProměnné = hodnotaProměnné;
int j = 15;
```


Proměnné

- proměnné je vhodné nazývat dle jejich významu

pojmenování proměnných:

alfanumerické znaky (písmena a čísla)

název proměnné začíná písmenem

malá/velká písmena se rozlišují

nepoužívejte diakritiku

nepoužívejte klíčová slova

podtržítko (víceslovné názvy se řeší oddělením slov pomocí podtržítka)

můžete používat tzv. **velbloudí styl** - první písmeno malé a každé počáteční písmeno dalšího slova bude velké

př. `x, X, maleCislo, velke_cislo`

Klíčová slova

- v nápovědě **C# Keywords**

```
abstract as base bool break byte case
catch class const continue decimal default
delegate do double else enum event
explicit extern false finally fixed float
for foreach get goto char checked if
implicit in int interface internal is let
lock long namespace new null object
operator out override params private
protected public readonly ref return sbyte
sealed set short sizeof stackalloc static
string struct switch this throw true try
typeof uint ulong unchecked unsafe ushort
using virtual void volatile while
```


Operátory

v nápovědě **C# Operators**

operátory se dělí na:

- primární
- unární - aplikují se na jeden operand
- binární
- operátory posunu
- operátory porovnávání
- operátory logické
- přiřazovací

Primární operátory

() operátor konverze typu, závorky

```
double x = 1234.5;
int a;
a = (int)x; // konverze double na int
```

. operátor

přístup k členské proměnné nebo metodě dané třídy

```
System.Console.WriteLine("Dobrý den!");
```


Primární operátory

[] operátor pole

- deklarace pole

```
int[] fib; // pole typu int
fib = new int[100]; // 100 prvků
```

- přístup k prvkům pole

```
fib[0] = fib[1] = 1;
  for( int i=2; i<100; ++i )
  {
     fib[i] = fib[i-1] + fib[i-2];
  }
```

Jaký index bude mít poslední prvek v poli *fib*?

Primární operátory

++ operátor inkrementace

vezme proměnnou C zvýší ji o jedničku

`C++;`

je ekvivalentní `C = C+1;`

nebo `C += 1;`

- má formu prefixovou (prefixový operátor se vyhodnocuje před přiřazením)

  `CC = ... ++C..;`

- a postfixovou (postfixový operátor se vyhodnotí až po přiřazení)

  `CC = .. C++ ..;`

Operátor inkrementace

```
using System;
    class MainClass {
      static void Main() {
        double x;
        x = 1.5;
        Console.WriteLine(++x);
        x = 1.5;
        Console.WriteLine(x++);
        Console.WriteLine(x);
      }
    }
```

výsledek?

- 2.5 1.5 2.5

Primární operátory

-- operátor dekrementace

zmenší hodnotu proměnné o 1

- **prefixový** - nejdříve se změní hodnota proměnné a pak se teprve vyhodnotí (s již změněnou hodnotou)
- **postfixový** - nejdříve se proměnná vyhodnotí (s původní hodnotou) a pak se teprve změní její hodnota

Primární operátory

operátor `new`

- vytvoření objektu

```
int[] fib; // pole typu int
fib = new int[100]; // 100 prvků
```

Operátory přiřazovací

= přiřazovací

+= k proměnné vlevo přičte hodnotu operandu vpravo

-= od proměnné vlevo odečte hodnotu operandu vpravo

***=** proměnnou vlevo vynásobí hodnotou operandu vpravo

/= proměnnou vlevo vydělí hodnotou operandu vpravo

%= do proměnné vlevo uloží zbytek po dělení původní hodnoty této proměnné hodnotou operandu vpravo

Unární operátory

mezi ně se řadí také **prefixové ++** a **--**

unární + nezmění znaménko operandu

unární - změna znaménka operandu

! operátor - logická negace

```
using System;
  class MainClass
  {
      static void Main()
      {
          Console.WriteLine(!true);
          Console.WriteLine(!false);
      }
  }
```

výsledek ?

False True

Unární operátory

- **operátor sizeof**

```
unsafe static void Main()
   {
    Console.WriteLine(„Velikost short je
  {0}.", sizeof(short));
    Console.WriteLine(„Velikost int je
  {0}.", sizeof(int));
    Console.WriteLine(„Velikost long je
  {0}.", sizeof(long));
   }
```

Velikost short je 2.

Velikost int je 4.

Velikost long je 8.

Binární operátory

a+b součet
a-b rozdíl
a*b násobení
a/b dělení (**celočíselné,** reálné)
i%k zbytek po dělení

```
int x = 5, y = 3;
x / y
1
x % y
2
```

Priorita operátorů

priorita operátorů řídí pořadí, v jakém jsou operátory uplatňovány ve výrazu

1. nejvyšší prioritu má unární operátor
2. (tady by bylo umocnění, ale C# nemá operátor pro mocninu)
3. následuje násobení a dělení
4. potom sečítání a odčítání

- výrazy se vyhodnocují odleva směrem doprava
- pokud potřebujete změnit prioritu operací, použijte kulaté závorky

Operátory porovnávání

méně než: `a < b`
méně nebo rovno: `a <= b`
větší než: `a > b`
větší nebo rovno: `a >= b`
není rovno: `a != b`
je rovno: `a == b`

Logické operátory

- ! (**negace**) - provede negaci hodnoty operandu vpravo a vrátí na dané místo vypočtenou hodnotu
- && (**logické a - konjunkce**) - provede konjunkci hodnoty operandů vlevo a vpravo a vrátí na dané místo vypočtenou hodnotu
- || (**logické nebo - disjunkce**) - provede disjunkci hodnot operandů vlevo a vpravo a vrátí na dané místo vypočtenou hodnotu
- ^ (**výlučné nebo**) - provede "výlučné nebo" hodnot operandů vlevo a vpravo a vrátí na dané místo vypočtenou hodnotu

Výlučné **nebo**

Hodnota levého operandu	Hodnota pravého operandu	Výsledek operace "výlučné logické nebo"
true	true	false
true	false	true
false	true	true
false	false	false

Přetížení operátoru: operátor +

```
class MainClass {
  static void Main() {
    Console.WriteLine(+5);         // unární plus
    Console.WriteLine(5 + 5);      // součet
    Console.WriteLine(5 + .5);     // součet
    Console.WriteLine("5" + "5"); // spojení řetězců
    Console.WriteLine(5.0 + "5"); // spojení řetězců
    // automatická konverze typu double na string
   }
  }
```

vypíše

5

10

5.5

55

55

Výjimky Exception

- aritmetické operace mohou vést k chybě
- ošetřit pomocí výjimky
  - OverflowException
  - DivideByZeroException

Příště: Řízení běhu programu