Operátory a výrazy

Úvod

S jedním ze základních operátorů už jsme se nevědomě setkali v předchozí kapitole. Byl to operátor přiřazení (`=`). V této kapitole se podíváme na další z nich a ukážeme si, jak s našimi proměnnými provádět některé operace.

Vymezení pojmu

Operátory jsou symboly, které specifikují, jakou z předdefinovaných operací provést na daném výrazu. Některé operátory jsou snadno čitelné, protože odpovídají operacím dobře známých z matematiky. Na začátku si představíme tzv. `binární operátory`.

Binární operátory

Mezi binární operátory řadíme `+` (sčítání), `-` (odčítání), `*` (násobení), `/` (dělení) a `%` (vyslovujeme „modulo“ - zbytek po celočíselném dělení). Opět si vše ukážeme na příkladu:

Binární operátory

```
// Pomocné proměnné
int a = 5;
int b = 2;

// Binární operace
int c = a + b; // => 7  (vrátí součet hodnot)
int d = a - b; // => 3  (vrátí rozdíl hodnot)
int e = a * b; // => 10 (vrátí součin hodnot)
int f = a / b; // => 2  (vrátí podíl hodnot, zbytek zahodí)
int g = a % b; // => 1  (vrátí zbytek po dělení)
```

Všechny tyto operace můžeme samozřejmě provádět i na jiných datových typech (pokud to umožňují, ale o tom více později) a dokonce i na různých datových typech mezi sebou. Musíme ale vždy dát pozor na to, aby výsledný datový typ výrazu korespondoval s datovým typem, do kterého se výsledek chystáme uložit. Příkladem budiž následující situace:

Podíl různých datových typů

```
// Pomocné proměnné
double a = 4.2;
int b = 2;

// Binární operace
double c = a / b; // => 2.1;
```

Z ukázky je patrné, že mezi sebou dělíme desetinné číslo typu `double` a celé číslo typu `int.` Protože je zde šance, že konečná hodnota toho výrazu bude opět desetinné číslo (a protože jeden z operandů je typu `double`), volíme pro proměnnou `c` opět datový typ `double`. Pokud bychom se i tak rozhodli výraz uložit do „intové“ proměnné, díky typové kontrole jazyka C# budeme upozorněni na naši chybu.

Rovněž se nyní nabízí otázka, co se stane, když mezi sebou budeme dělit například čísla typu `float` a `double`. Bude výsledkem `float` nebo `double`? Tato problematika je předmětem tzv. implicitních konverzí, které jsme probírali v předchozí kapitole.

Je také důležité si uvědomit, že ne všechny operace mohou být provedeny na všech datových typech. Dobrým příkladem je třeba datový typ `string`. Jednotlivé řetězce můžeme mezi sebou skládat pomocí operátoru `+`:

Podíl různých datových typů

```
// Pomocné proměnné
string jmeno = "Petr";
string prijmeni = "Novák";

// Spojení řetezců
string celeJmeno = jmeno + " " + prijmeni; // => Petr Novák
```

Použití ostatních binárních operátorů (`-`, `*`, `/` a `%`) by v tomto případě skončilo chybou. Pokud se nám podaří zapsat jakoukoliv neplatnou operaci, budeme vždy varování chybovým hlášením. Při zapsání více operací na ráz se postupuje stejným způsobem, který známe z matematiky. Násobení a dělení (popřípadě modulo) má přednost před sčítáním a odčítáním, neurčí-li kulaté závorky jinak. Pokud chceme použít více vnořených závorek v sobě, používáme pouze kulaté závorky. Hranaté, špičaté a složené závorky využijeme k jiným účelům. Pokud použijeme více operátorů stejné úrovně za sebou, postupuje se zleva doprava:

Kombinace operátorů

```
// Násobení, dělení a modulo má
// přednost před sčítáním a odčítáním
int a = 2 * 2 + 5; // => 4 + 5 = 9
```

```
// Kulaté závorky mohou toto
// pravidlo změnit
int b = 2 * (2 + 5); // => 2 * 7 = 14

// Více operátorů stejné úrovně –
// postupujeme zleva doprava
int c = 2 * 8 / 4; // => 16 / 4 = 4
```

Unární operátory

Unární operátory (česky jednočlenné) vyžadují pouze jeden operand (proměnnou) pro své fungování. Jedním takovým operátorem je i negace (obecný zápis `-x`, kde `x` je název proměnné):

Podíl různých datových typů

```
// Proměnná věk
int vek = 5;

// Negace proměnné věk
int negaceVek = -vek; // => -5
```

Mezi unární operátory patří celá řada dalších zástupců. My si pro jednoduchost a srozumitelnost zatím vystačíme pouze s tímto.

Operátory přiřazení

Jak bylo řečeno v úvodu, operátor přiřazení (`=`) je nám již známý. Pro jistotu jen znovu zopakuji, že tento operátor přiřadí pravou část výrazu do části levé. Musí však být vždy zachováno pravidlo, že výsledné datové typy obou částí jsou shodné. Zkušenější programátoři mi jistě odpustí, protože toto není díky dědičnosti a implicitním konverzím úplná pravda. Pro větší jednoduchost ale opět zamlčíme některá fakta.

Do této skupiny mimo jiné spadají i operátory složeného přiřazení, mezi které řadíme `+=`, `-=`, `*=`, `/=` a `%=`. Tyto operátory zjednodušují přiřazování hodnot do proměnných. Jejich užití si osvětlíme na následující ukázce:

Operátory složeného přiřazení (1)

```
// Pomocná proměnná
int a = 2;
```

```
// Operátory složeného přiřazení
a += 1; // => 3 (přičteme jedničku)
a -= 1; // => 2 (odečteme jedničku a vrátíme se na původní hodnotu)
a *= 4; // => 8
a /= 2; // => 4
a %= 3; // => 1
```

Složené operátory jsou kratší variantou pro klasický operátor přiřazení spojený s binární operací:

Operátory přiřazení (2)

```
// Pomocná proměnná
int a = 2;

// Operátory složeného přiřazení přepsané na
// své ekvivalenty pomocí operátoru přiřazení
// a binárního operátoru
a = a + 1; // => 3
a = a - 1; // => 2
a = a * 4; // => 8
a = a / 2; // => 4
a = a % 3; // => 1
```

Důležitost typového systému

V minulé kapitole jsme si ukázali, že číslici `10` lze jednou uložit jako textový řetězec a poté i jako číslo. Pro připomenutí znovu uvedu příklad:

Způsoby uložení

```
// Číslice 10 jako číslo
int a = 10;

// Číslice 10 jako text
string b = "10";
```

S naší novou znalostí binárního operátoru `+` si nyní můžeme ukázat, jak datový typ kompletně změní výsledek operace součtu:

Podíl různých datových typů

```
// Číslice 10 jako číslo
int a = 10;

// Číslice 10 jako text
string b = "10";

// Součet "číslic"
int x = a + a; // => 20
string y = b + b; // => "1010"
```

Toto je pouze jeden příklad z mnoha. V budoucnosti zjistíme, že datové typy budou ovlivňovat téměř každou část našeho kódu.

Shrnutí

V této kapitole jsme se seznámili s operátory. Pro naše další účely nebude důležité znát jejich přesné rozdělení, ale pouze znát jejich význam a reálné využití.

Revision #1
Created 2025-05-21 06:46:46 UTC by Magdalena Dobešová
Updated 2025-05-21 06:54:11 UTC by Magdalena Dobešová