X36PJC
7. přednáška – 2. část

Operátory, výrazy

Operátory

C/C++ operátory:

- Aritmetické: unární **-, +, -, *, /, % (modulo),**
- Bitové: **~, &, |, ^, <<, >>,**
- Logické: **!, &&, ||,**
- Relační: **<, <=, >, >=, ==, !=,**
- Přiřazovací: **+=, -=, %=, >>=, &=, …,**
- Ternární: **? :**
- Inkrement/dekrement: **++, --**
- Volání funkce, indexace: **(), []**
- Přístup ke složkám struktury: **., ->**
- Reference a dereference: **&, ***

Operátory

Aritmetické operátory:

- zápis podobný jako v jiných pgm. jazycích,
- zbytek po dělení - **%,**
- typ výsledku je určen typem operandů,
- automatické konverze datových typů před provedením operace.

Příklad:

```
double x;
x     = 2   / 4;    // x = 0.0, proc ?
x     = 2.0 / 4;    // x = 0.5
x     = 2   / 4.0f; // x = 0.5
```

Operátory

Bitové operátory:

- **|** **or,**
- **&** **and,**
- **^** **xor,**
- **~** **bitová negace,**
- **>>, <<** **aritmetický posuv vpravo / vlevo.**
- Jak zařídit bitový posuv (Java operátory **<<< a >>>)?** **Unsigned operandem.**

Příklad:

```
int x;          // 37 = 0010 0101
x = 37 | 94;    // 94 = 0101 1110
                // x  = 0111 1111 => 127
```

Operátory

Příklady použití bitových operátorů:

- Vytvoření masky, kde je nastaven pouze i-tý bit:
  mask = 1 << i
- Vytvoření masky, kde je nulován pouze i-tý bit:
  mask = ~(1 << i)
- Nastavení i-tého bitu na 1:
  val = val | (1 << i)
- Nastavení i-tého bitu na 0:
  val = val & ~(1 << i)
- Překlopení i-tého bitu:
  val = val ^ (1 << i)
- Test, zda je i-tý bit nastaven:
  (val & (1 << i)) != 0

Operátory

Logické operátory:

- **||** **or,**
- **&&** **and,**
- **!** **logická negace,**
- výsledkem je hodnota **0 (false) nebo 1 (true),**
- zkrácené vyhodnocení (ukončí se v okamžiku, kdy je jasný výsledek).

Příklad:

```
x = 37 || 94;  // x = 1 (true)
x = 37 && 94;  // x = 1 (true)
x = ! 37;      // x = 0 (false)
x = x && delAll ( "C:\\" ); // nesmaze
```

Operátory

Relační operátory:

- **<, <=, …,**
- výsledkem je hodnota **0 (false) nebo 1 (true),**
- pozor na asociativitu.

Příklad:

```
x = 5;
if ( 10 < x < 30 ) doJob ();        // does
                                    // ( ( 10 < x ) < 30 )
if ( a == b == c == d ) …   // ((( a == b ) == c ) == d )
```

Operátory

Přiřazovací operátory:

- **=, +=, -=, …,**
- pravě asociativní, lze seskupovat,
- vedlejší efekt (zápis do paměti) není serializovaný.

Příklad:

```
x = y = z = 0; // x = ( y = ( z = 0 )));
x += 10;       // x = x + 10
x -= 20 + 30;  // x = x - ( 20 + 30 );
x *= x *= 20;  // nedefinovano
```

Operátory

Ternární operátor:

- **podmínka ? hod_pravda : hod_nepravda**
- vyhodnotí právě jeden z výrazů pravda / nepravda,
- výrazy větví pravda a nepravda musejí mít stejný (konvertovatelný) typ.

Příklad:

```
max2 = ( x > y ) ? x : y;
absx = x >= 0 ? x : -x;
cout << "X je " << (X > 0 ? "kladne" : "nekladne") << endl;
cout << "X je " << (X > 0 ? "kladne" : X ) << endl;
```

Operátory

Další operátory:

- **++, --** **pre/post inkrement/dekrement,**
- **,** **operátor "zapomenutí",**
- ***,&** **(jako unární) dereference / reference,**
- **.,->** **přístup k složkám třídy / struktury**
- **(),[] (jako postfixové) volání funkce, indexace.**

Příklad:

```
int a = 4, b;
b = a ++;      // b = 4, a = 5
b = ++ a;      // b = 6, a = 6
b = a ++ ++;   // !!
```

Operátory

Pr.	Operátory	Asociativita
1	`()  []  ->  .`	zleva doprava
2	`!  ~  ++  --  +  -  (typ)  *  &  sizeof`	zprava doleva
3	`*  /  %`	zleva doprava
4	`+  -`	zleva doprava
5	`<<  >>`	zleva doprava
6	`<  >  >=  <=`	zleva doprava
7	`==  !=`	zleva doprava
8	`&`	zleva doprava
9	`^`	zleva doprava
10	`|`	zleva doprava
11	`&&`	zleva doprava
12	`||`	zleva doprava
13	`?:`	zleva doprava
14	`=  +=  -=  *=  /=  %=  >>=  <<=  &=  |=  ^=`	zprava doleva
15	`,`	zleva doprava

Výrazy

- Výraz je vyhodnocován podle priorit operátorů a jejich asociativity.
- Změna priority či asociativity pomocí závorek.
- Pořadí vyhodnocení není garantováno, k dispozici pro optimalizaci.
- Pořadí je definované pro ternární operátor a čárku.
- Zkrácené vyhodnocení logických operátorů.
- Pořadí vyhodnocení může být důležité, pokud má podvýraz vedlejší efekt (volání funkce, přiřazení, **++,...).**
- V C/C++ lze zapsat nedefinované výrazy.

Výrazy

l-value:

- výraz, který může stát na levé straně operátoru =,
- má paměťovou reprezentaci (je kam uložit výsledek),
- lze na něj vytvořit ukazatel či referenci.

r-value:

- výraz, který může stát na pravé straně operátoru =,
- má hodnotu, ale nemá paměťovou reprezentaci,
- např. může existovat pouze v registru CPU během výpočtu, pak je zapomenut.

Výrazy

Příklady l-value a r-value:

int a, *b = &a, *c[5], & d = a;

l-value	r-value
a	**a + 5**
b	**2 * a**
c[3]	**&a**
d	**&b**
***b**	**c**
***c**	**&c[3]**
***(&a)**	**a ++**
***(b+1)**	**a > 0 ? *b : d**

Výrazy

Pozor na prioritu a asociativitu:

```
int a, b, c, *d = &a;

c = a + b >> 1;    // c = (a + b) >> 1

c = a & 2 == 2;    // c = a & (2 == 2)

c = 3 * a / 4 * b; // c = (3 * a / 4) * b

c = *d ++;         // c = *(d ++)
```

Výrazy

Nedefinované výrazy:

- Není definováno v jakém pořadí jsou vyhodnoceny podvýrazy.
- Není definováno, kdy se do paměti zapíše výsledek, pokud má operátor vedlejší efekt (**++, =, +=, ...**).
- Všechny vedlejší efekty se uplatní nejpozději po skončení příkazu, operátorech **||, &&, ?: a ,**.

Příklad:

```
int a = 1, c;
c = a++ + a++; // a = 2, 3 ?
               // c = 2, 3 ?
```

Děkuji Vám za pozornost