Výrazy, dokončení

Petr Šaloun

katedra informatiky FEI VŠB-TU Ostrava

10. října 2005

výraz

složen z **operátorů** a **operandů**
C++: každý operátor přetížit kromě
?:, ::, ., .*, sizeof , typeid , const_cast ,
dynamic_cast , reinterpret_cast , static_cast

operátor – funkce – kontext, například <<

Přiřazujeme výsledek Pythagorova věta

$$c = \sqrt{a^2 + b^2}$$

```
c = sqrt(a*a + b*b);
```

Operátor přiřazení

```
int a, b, c;
a = b = c = -1;
```

rvalue – **r-hodnota**, *hodnotový výraz*
lvalue – **l-hodnota**, *adresový výraz*
nejednoznačné:

```
cc = cc++ * 2;
a[i] = i++;
```

Aritmetické výrazy

+, −, *, /, %

```
123 * 456 = 56088
295 /   2 =   147
295 %   2 =     1

nyni pozor:
     20000 *       20001 = 400020000
 400020000 *          10 = -294767296
```

Aritmetické výrazy – výpis 1

```
#include <iostream>      // vstupy a vystupy
#include <iomanip>  // formatovani vystupu

using namespace std;

int main() {
 int o1 = 123, o2 = 456, o3 = 295, v1, v2, v3;
 int c1 = 20000, c2 = 20001, vc;
 v1 = o1 * o2;
 v2 = o3 / 2;
 v3 = o3 % 2;

 cout << o1 << " * " << o2 << " = " << setw(5) << v1 <
 cout << o3 << " /   2 = " << setw(5) << v2 << endl;
 cout << o3 << " %   2 = " << setw(5) << v3 << endl;
```

Aritmetické výrazy – výpis 2

```
  vc = c1 * c2;
  cout << endl << "nyni pozor:" << endl;
  cout << setw(10) << c1 << " * " << setw(10) << c2 <<
  cout << setw(10) << vc << " * " << setw(10) << 10 <<
  return 0;
} // int main()
```

smíšené aritmetické výrazy

```
int main() {
 int i, j;
 double r, x;

 j = i = 5;
 j *= i;
 r = j / 3;
 x = j * 3;
 cout << "i=" << i << " j=" << j << " r=" << r
          << " x=" << x << endl;

 return 0;
} // int main()
```

```
i=5 j=25 r=8 x=75
```

Logické hodnoty a operátory

C++ bool, true a false
C celá čísla 1, 0
&&, ||, ! – *konjunkce*, *disjunkce*, *negace*

A	B	!A	A && B	A || B
false	false	true	false	false
false	true	true	false	true
true	false	false	false	true
true	true	false	true	true

Relační operátory

<, >, <=, >=, ==, !=

menší než, větší než, menší nebo rovno, větší nebo rovno, rovno a nerovno

Bitové operátory

pouze s celočíselnými hodnotami

<<, >>, &, |, ~, ^

posun vlevo, *posun vpravo*, bitová *konjunkce*, bitová *disjunkce*, bitová *negace*, bitová *nonekvivalence*

b1	b2	~b1	b1 & b2	b1 | b2	b1 ^ b2
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

Bitový posun, MSB, LSB

– unsigned, celočíselné dělení (násobení) dvěma,

– aritmetický posun, znaménkový bit.

Bitová konjunkce &, *disjunkce* |, a *nonekvivalence* ^ (and, or, xor) provádí příslušnou binární operaci s každým párem odpovídajících si bitů celočíselných operandů.

Bitová negace ~ je unární, bitový doplněk.

ve 32bitovém prostředí:

```
  1 << 1 =      2   0x2
  1 << 7 =    128   0x80
 -1 >> 1 =     -1   0xffffffff
512 >> 8 =      2   0x2
 13  & 6 =      4   0x4
 13  | 6 =     15   0xf
 13  ^ 6 =     11   0xb
  2  & 1 =      0     0
  2  | 1 =      3   0x3
  2  ^ 1 =      3   0x3
```

op-bit01.cpp část 1

```
#include <iomanip>  // formatovani vystupu

using namespace std;

int main() {
 int vlevo1 = 1 << 1;
 int vlevo7 = 1 << 7;
 cout.setf(ios::showbase); // ukaze zaklad ciselne sou
 cout << "  1 << 1 = " << dec << setw(6) << vlevo1 <<
 cout << "  1 << 7 = " << dec << setw(6) << vlevo7 <<

 int vpravo1 = -1 >> 1;
 int vpravo8 = 512 >> 8;
 cout << " -1 >> 1 = " << dec << setw(6) << vpravo1 <<
 cout << "512 >> 8 = " << dec << setw(6) << vpravo8 <<
```

op-bit01.cpp část 2

```
int k = 13 & 6;
int d = 13 | 6;
int non = 13 ^ 6;
cout << " 13  & 6 = " << dec << setw(6) << k << hex <
cout << " 13  | 6 = " << dec << setw(6) << d << hex <
cout << " 13  ^ 6 = " << dec << setw(6) << non << hex

k = 2 & 1;
d = 2 | 1;
non = 2 ^ 1;
cout << "  2  & 1 = " << dec << setw(6) << k << hex <
cout << "  2  | 1 = " << dec << setw(6) << d << hex <
cout << "  2  ^ 1 = " << dec << setw(6) << non << hex
return 0;
```

int-size.cpp

```
#include <iostream>      // vstupy a vystupy

using namespace std;

int main() {
 unsigned int ui = ~0;
 int i = 1;
 while (ui >>= 1)
   i++;
 cout << "prekladac pouziva " << i << "bitovou repreze
 return 0;
} // int main()
```

```
prekladac pouziva 32bitovou reprezentaci celeho cisla
```

Adresový operátor

& je unární

```
promenna i=123 je umistena od adresy: 0012FF7C
```

```
#include <iostream>

using namespace std;

int main() {
 int i = 123, *pi;

 pi = &i;
 cout << "promenna i=" << i << " je umistena od adresy
 return 0;
} // int main()
```

Reference – odkaz

synonymum, alias

```
int  i     = 0; //celociselna promenna
int & oi = i; // oi je vytvoreno jako odkaz na i
oi         = 2; // umisti 2 do promenne i (pres odkaz oi
```

Operátor čárka

postupné vyhodnocení, nejnižší priorita, zleva doprava.

Přetypování výrazu

(typ), const_cast , static_cast , dynamic_cast, reinterpret_cast
(typ) vyraz
r = j / 3;
r = (double) j / 3;

Podmíněný operátor, op-cond.cpp

ternární

```
#include <iostream>

using namespace std;

int main() {
 int i, abs_i;

 cout << endl << "Zadej cele cislo: ";
 cin >> i;

 abs_i = (i < 0) ? -i : i;
 cout << "abs(" << i << ") = " << abs_i << endl;
 return 0;
} // int main()
```

```
    Zadej cele cislo: 123
```

Správa paměti, new-del.cpp část 1

C++ new a delete.
new alokuje objekt – **kostruktor**

```
#include <iostream>
using namespace std;

int main() {
  int *ptr = new int(3);
  *ptr += 7;
  cout << "*ptr = " << *ptr << endl;
  delete ptr;
```

Správa paměti, new-del.cpp část 2

```
  int pocet;
  cout << "Kolikaprvkove pole chces vytvorit?:";
  cin >> pocet;

  double *pole = new double[pocet];
  int i = 0;
  while (i < pocet) {
          pole[i] = i + (double) i / 100;
          cout << pole[i] << endl;
          i++;
  } // while (i < pocet)
  delete [] pole;
  return 0;
} // int main()
```

Správa paměti, new-del.cpp výstup

```
*ptr = 10
Kolikaprvkove pole chces vytvorit?:5
0
1.01
2.02
3.03
4.04
```

Výrazový příkaz, příkaz

výraz ukončený ;
Prázdný příkaz
;
Blok skupina příkazů mezi { a }
složený příkaz
lokální deklarace a definice, *lokální proměnné*,
paměťová třída auto

Oblast platnosti identifikátoru

- Na úrovni souboru začíná platnost deklarace místem deklarace a končí na konci souboru.
- Deklarace na úrovni argumentu funkce má rozsah od místa deklarace argumentu v rámci definice funkce až do ukončení vnějšího bloku definice funkce. Pokud se nejedná o definici funkce, končí rozsah deklarace argumentu s deklarací funkce.
- V rámci bloku je deklarace platná do konce bloku.

Prostor jmen

řeší problémy s případným konfliktem v oblasti identifikátorů.
using namespace std;

vlastní:
namespace identifikator seznam deklaraci
operátor čtyřtečka
::

prostor_jmen::identifikator

scope-jmena.cpp

ží

```
const char *id = "soubor scope_jmena.cpp";
  // áíglobln konstanta

namespace CPP_pro_zelenace {
  const char * id = "Saloun. C++ pro zelenace.";
  // konstanta ve éjmennm prostoru CPP_pro_zelenace
}
int main() {
  const char *id = "telo funkce main";
  // áílokln konstanta ve funkci main()
  std::cout << id << std::endl; // áílokln
  std::cout << ::id << std::endl; // áíglobln
  std::cout << "Dobra kniha: " << CPP_pro_zelenace::id
            << std::endl;
  return 0;
```

scope-jmena.cpp výstup

```
telo funkce main
soubor scope_jmena.cpp
Dobra kniha: Saloun. C++ pro zelenace.
```