Operatori C++

Toate articolele | Articolele lui **Candale Silviu**

 **Candale Silviu (silviu)** • **22.11.2020**

Etichete: Dificultate: începător C++

Computerele prelucrează date! Le citesc de la tastatură, le memorează în variabile (sau constante), le afișează pe ecran. Și mai fac cu ele diverse operații. Noi suntem obișnuiți să facem operații aritmetice (adunări, scăderi, etc.), dar în C++ există multe alte operații.

Introducere

O **operație** este alcătuită din **operanzi** și **operator**. Operanzii reprezintă datele cu care se fac operațiile, iar operatorul este simbolul care stabilește ce operație se face cu operanzii. Din punct de vedere a numărului de operanzi, operațiile (operatorii) pot fi:

- **unare** – se aplică la un singur operand (de exemplu `-7`, operația de schimbare a semnului unui număr);
- **binare** – se aplică la doi operanzi (de exemplu adunarea numerelor, `2+5`);
- **ternare** – se aplică la trei operanzi. Există un singur operator ternar în C++, **operatorul condițional** și va fi analizat mai jos.

Operanzii pot fi variabile, constante, literali, rezultatele unor funcții, rezultatele altor operații. O operație care are ca operanzi alte operații se numește **expresie**.

> **Notă:** Fiecare operație C++ are un rezultat!

Acest articol analizează o parte a operatorilor C++, cei mai frecvent utilizați:

Aceștia sunt: `+`, `-`, `*`, `/`, `%`. În exemplele de mai jos, considerăm variabilele:

- `N = 11` și `M = 3` de tip `int`
- `X = 11` și `Y = -3.5` de tip `double`.

Operatorii aritmetici unari

În C++ există operațiile unare `+` și `-`. `+` returnează valoarea operandului, iar `-` returnează valoarea operandului cu semn schimbat.

Exemple

- `+ X = 11`
- `- Y = 3`
- `- + N = -11`

Operatorii aritmetici binari

- `+` : adunarea a două numere
- `-` : scăderea a două numere
- `*` : înmulțirea a două numere
- `/` : împărțirea a două numere
- `%` : restul împărțirii a două numere întregi (**modulo**)
- În C++ nu există un operator pentru ridicarea la putere!

Adunarea, scăderea și înmulțirea se comportă conforma așteptărilor, ca la matematică. Operația de împărțire și operația modulo necesită niște explicații suplimentare.

Împărţirea întreagă şi împărţirea zecimală

Operația de împărțire are două moduri de lucru, în funcție de tipul operanzilor.

- Dacă operanzii sunt de tip întreg (`int`, `short`, `char`, etc.), se va realiza împărțirea întreagă, iar rezultatul operației `/` este câtul împărțirii întregi.
- Dacă operanzii sunt de tip real (`float`, `double`, `long double`), se va realiza împărțirea zecimală, iar rezultatul operației `/` este rezultatul acestei împărțiri, “cu virgulă”.

Exemple

- `N / M = 3`
- `X / Y = -3.14286`
- `X / 2.0 = 5.5`
- `M / 2 = 1`
- `M / 2.0 = 1.5`

Ultima împărțire este deosebită. Cei doi operanzi au tipuri diferite: `M = 3` este de tip `int`, iar `2.0` este de tip `double`. Aici intervine operația de **conversie implicită**: în mod automat, operandul `M` se consideră ca fiind de tip `double`, împărțirea este împărțire reală și are rezultatul `1.5`.

Operatorul modulo `%`

`N % M = 2` : restul împărțirii lui `11` la `3` este `2`
`30 % 10 = 0`

Operatorul modulo este util în multe situații. El poate fi utilizat pentru a afla ultima cifră a unui număr natural: ultima cifră a lui `276` este `276 % 10` adică `6`, sau pentru a verifica dacă un număr `N` este divizor al lui `M`. În caz afirmativ, `M % N` este `0`.

Observații suplimentare

- nu se poate face împărțire la `0`.
- dacă cel puțin un operand al împărțirii întregi sau al operației modulo este negativ, rezultatul operației depinde de versiunea compilatorului C++ folosit. O variantă de evaluare, utilizată în unele compilatoare, este efectuarea operației cu operanzii în valoare absolută și apoi aplicarea regulii semnelor.

Operatorii relaționali

Sunt: `<`, `>`, `<=`, `>=`, `==`, `!=` .

Un operatori relațional stabilește dacă între două numere (operanzii) are loc o anumită relație. Rezultatul acestei operații este **adevărat** sau **fals**. Rezultatul operațiilor relaționale poate fi `0` sau `1`:

- rezultatul este `1` dacă relația este adevărată
- rezultatul este `0` dacă relația este falsă

Fie `N = 11` și `M = 3`. Operațiile relaționale sunt:

- operații de comparare (comparații)
  - mai mic `<`; `N < M` este fals, adică `0`
  - mai mare `>`; `N > M` este adevărat, adică `1`
  - mai mic sau egal `<=`; `M <= N` este `1`
  - mai mare sau egal `>=`; `M >= N` este `0`
- operația de egalitate `==` ; `N == M` este fals, adică `0`
- operația de neegalitate (diferit, not egal) `!=`; `N != M` este adevărat, adică `1`.

> Una dintre cele mai frecvente erori este folosirea pentru operația de egalitate a operatorului `=`, în loc de `==`. Operatorul `=` reprezintă operația de atribuire!

Operatorii logici

Sunt: `!`, `||`, `&&`.

Operatorii logici au operanzi de tip valori de adevăr și rezultat valori de adevăr. Istoric, operațiile logice sunt legate de numele matematicianului englez George Boole, cel care a pus bazele acestei ramuri a matematicii și a inventat **algebra booleană** și calculul propozițional.

În C++, operatorii logici pot fi aplicați oricăror valori numerice, și au ca rezultat una din valorile `0` sau `1`. În exemplele de mai jos vom folosi literalii `true` și `false`, de tip `bool`.

- `! true` este `false`. Orice valoare nenulă negată devine `0`.
- `! false` este `true`. `0` negat devine `1`.

Disjuncția: ||

- `false || false → false`
- `false || true → true`
- `true || false → true`
- `true || true → true`

Conjuncția: &&

- `false && false → false`
- `false && true → false`
- `true && false → false`
- `true && true → true`

Legile lui De Morgan

Fie `p` și `q` două valori booleene (pot fi rezultatele unor expresii, de exemplu). Atunci:

- `!(p && q) == !p || !q`
- `!(p || q) == !p && !q`

Să luăm ca exemplu apartenența unei valori la un interval:

- `x∈[a,b]`
- expresia echivalentă C++ este `x ≥ a && x ≤ b`
- conform legilor lui De Morgan, prin negare obținem: `!(x ≥ a) || !(x ≤ b)`
- deoarece `!(x ≥ a)` este echivalent cu `x<a`, obținem:
- `x < a || x > b`
- folosind intervalele obținem: `x ∈ (-∞,a)∪(b,+∞)`, adică `x∉[a,b]`

Operatorul de atribuire: =

Atribuirea este operația prin care **o variabilă primește valoarea unei expresii**:

`variabila = expresie`

Expresia poate avea orice fel de rezultat, dacă tipul său acesta este identic cu al variabilei, sau poate fi convertit la tipul variabilei. În cazul tipurile întregi, reale, `bool`, oricare dintre acestea poate fi convertit la la oricare altul, eventual cu pierderea unor date.

Exemple:

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

```
    double x , y; // valori aleatorii
    n = 5; // valoare lui n devine 5
    cout << n << endl;
    m = n + 2; // valoare lui m devine 7
    cout << m << endl;
    n = n + 3; // valoarea lui n devine 5 + 3, adica 8
    cout << n << endl;
    x = m / 5; // valoarea lui x devine 8 / 5, adica 1. ATENTIE! este impartire intreaga
    cout << x << endl;
    y = 5; // valoarea lui y devine 5, de tip double. Are loc conversia lui 5 de tip int la dou
    cout << y << endl;
    x = m / y; // valoarea lui x devine 1.4, deoarece impartirea este zecimala. Are loc convers
    cout << x << endl;
    return 0;
}
```

> Atribuirea este o operație, deci are rezultat! Rezultatul operației de atribuire este chiar variabila care primește valoare.

> Nu confundați operația de atribuire (`=`) cu operația de egalitate `==`.

Este posibilă și realizarea unor atribuiri multiple, ca mai jos:

```
int a , b, c;
a = b = c = 10;
```

Toate variabilele vor primi valoarea `10`.

Următoarea atribuire este mai interesantă:

```
n = n + 4;
```

Ea se efectuează astfel (să considerăm, ca exemplu, că valoarea inițială a lui `n` este `5`):

- mai întâi se calculează expresia din dreapta, în care se folosește valoarea curentă a lui `n`: `n + 4` este `5 + 4` adică `9`
- valoarea expresiei din dreapta se atribuie variabilei din stânga, deci `n` devine `9`.

Nota: În membru stâng al unei atribuiri poate fi nu doar o variabilă, ci o expresie de tip **lvalue**. Prin **lvalue** se înțelege **left value**, adică tocmai o expresie ce poate "în stânga" unei atribuiri. Variabilele sunt expresii *lvalue*, dar există și altfel de expresii, despre care vom vorbi mai târziu, care sunt *lvalue*.

Operatorii de atribuire compusă

Sunt: `+=`, `-=`, `*=`, `/=`, `%=`, `>>=`, `<<=`, `&=`, `^=`, `|=`.

În programare sunt foarte frecvente atribuirile de forma:

```
x = x * 5;
```

compusă:

`var OP= expresie`, echivalentă cu `var = var OP expresie`

Astfel, atribuirea `x = x * 5` este echivalentă cu `x *= 5`.

Operatorii ++, --

Se numesc operatori de de incrementare (`++`) și decrementare (`--`).

Prin incrementarea unei variabile se înțelege mărirea valorii sale cu `1`. Similar, prin decrementarea unei variabilă se înțelege micșorarea valorii sale cu `1`.

Operația de **incrementare** a variabilei `x` poate fi:

- **postincrementare**: `x ++`. Efectul expresiei este mărirea valorii lui `x` cu `1`, iar rezultatul expresiei este valoarea inițială a lui `x`.
- **preincrementare**: `++ x`. Efectul expresiei este mărirea valorii lui `x` cu `1`, iar rezultatul expresiei este valoarea mărită a lui `x`.

Exemplu pentru postincrementare:

```
int x = 5 , y = 10;
y = x ++; // y primeste valoare lui (x++), adica valoarea initiala a lui x
cout << x << " " << y; // 6 5
```

Exemplu pentru preincrementare:

```
int x = 5 , y = 10;
y = ++ x; // y primeste valoare lui (++x), adica valoarea marita a lui x
cout << x << " " << y; // 6 6
```

Operația de **decrementare** a variabilei `x` poate fi:

- **postdecrementare**: `x --`. Efectul expresiei este micșorarea valorii lui `x` cu `1`, iar rezultatul expresiei este valoarea inițială a lui `x`.
- **predecrementare**: `-- x`. Efectul expresiei este micșorarea valorii lui `x` cu `1`, iar rezultatul expresiei este valoarea micșorată a lui `x`.

Operatorul condițional ?

Operatorul condițional este singurul operator ternar (cu trei operanzi) din C++. Sintaxa lui este:

`expresie1 ? expresie2 : expresie3`

și se evaluează astfel:

- se evaluează `expresie1`
- dacă rezultatul lui `expresie1` este nenul (adevărat), se evaluează `expresie2` și rezultatul acestei expresii va fi rezultatul operației `?`

`expresie2` și `expresie3` trebuie să aibă rezultate de același tip, sau de tipuri compatibile

Exemplu:

```
int x;
cin >> x;
cout << (x % 2 == 0? "par" : "impar");
```

Operatorul virgulă ,

În anumite situații, regulile de sintaxă ale limbajului C++ solicită prezența unei singure operații, dar logica programului cere prezența mai multor operații. Acestea pot fi grupate cu ajutorul operatorului `,`. Sintaxa acestei operații este;

`expresie1 , expresie2`

Modul de evaluare este:

- se evaluează mai întâi `expresie1`, apoi `expresie2` – important, dacă în `expresie2` apar variabile care se modifică în `expresie1`
- rezultatul operației este rezultatul lui `expresie2`

Exemple:

```
int x , y , z;
x = 1 , y = 2 , z = 3;
x ++, y = x + 2, z -= x; // este semnificativa ordinea in care s-au evaluat cele trei expresii
cout << x << " " << y << " " << z; // 2 4 1
```

Operatorii pe biți

Sunt: `&`, `|`, `^`, `~`, `<<`, `>>`.

Operatorii pe biți reprezintă o temă avansată de programare. Ei permit manipularea directă și foarte rapidă a biților care formează reprezentarea în memorie a unei date. Vezi acest articol!

Operatorul de conversie explicită

În anumite situații trebuie să considerăm o expresie de un anumit tip ca fiind de alt tip. Acest lucru poate fi realizat prin operatorul de conversie:

`(tip_nou) expresie`

Exemple:

```
int x = 2;
cout << 7 / x << endl; // 3 - este impartire intreaga
cout << 7 / (double) x; // 3.5 - este impartire zecimala
```

```
cout << p - 32 << endl; // 65
cout << (char)(p - 32); // A - carcaterul cu codul ASCII 65
```

Operatorul sizeof

Operatorul `sizeof` este un operator unar care se aplică la un tip de date sau la o expresie. Rezultatul său este numărul de octeți pe care îi ocupă o dată de acel tip, respectiv rezultatul expresiei.

Exemple

```
cout << sizeof(double) << endl; // 8: o data de tip double ocupa 8 octeti
cout << sizeof(3 + 5) << endl; // 4: 3 + 5 este de tip int; o data de tip int ocupa 4 octeti
```

Alți operatori

Limbajul C++ conține și alți operatori, dintre care:

- `( )` – modificarea priorității unei operații, apel de funcție
- `[ ]` – indexarea unui tablou
- `.`, `->` – acces la membrii unei structuri
- `&`, `*` – referențiere (determinarea adresei unei variabile), dereferențiere (accesare variabilei de la o adresă)
- `new`, `delete` – alocare și dealocarea memoriei
- `<<`, `>>` – inserare și extragere din stream
- `::` operatorul de acces/rezoluție

Prioritatea operatorilor

Prioritatea operatorilor stabilește ordinea în care se evaluează o expresie care conține mai mulți operatori, de diverse feluri – ordinea în care se efectuează operațiile.

Asocierea operatorilor stabilește ordinea în care se evaluează o expresie ce conține mai mulți operatori cu aceeași prioritate. Poate fi **de la stânga la dreapta** sau **de la dreapta la stânga**.

Atât prioritatea, cât și asocierea operatorilor poate fi modificată folosind paranteze rotunde `()`

Pentru operatorii prezentați mai sus, prioritatea este următoarea:

Nivelul 1 de prioritate. Asociere: *stânga dreapta*

- operatorul de rezoluție `::`

Nivelul 2 de prioritate. Asociere: *stânga dreapta*

- postincrementarea / postdecrementare `++ --`
- accesul la elementele unui tablou `[]`
- modificarea priorității unei operații, apel de functie `()`

- preincrementarea / predecrementare `++ --`
- negarea pe biți `~`, negarea logică `!`
- operatorii aritmetici unari `+ -`
- referențierea / dereferențierea `& *`
- alocarea / dealocarea memoriei `new delete`
- operatorul `sizeof`
- operatorul de conversie explicită

Nivelul 4 de prioritate. Asociere: *stânga dreapta*

- pointeri la membri unei structuri `.* ->*`

Nivelul 5 de prioritate. Asociere: *stânga dreapta*

- operații aritmetice multiplicative `* / %`

Nivelul 6 de prioritate. Asociere: *stânga dreapta*

- operații aritmetice aditive `+ -`

*Nivelul 7 de prioritate. Asociere: *stânga dreapta*

- deplasare pe biți `<<`, `>>`

Nivelul 8 de prioritate. Asociere: *stânga dreapta*

- operatorii relaționali de comparare `< > <= >=`

Nivelul 9 de prioritate. Asociere: *stânga dreapta*

- operatorii relaționali de egalitate / neegalitate `== !=`

Nivelul 10 de prioritate. Asociere: *stânga dreapta*

- ȘI pe biți `&`

Nivelul 11 de prioritate. Asociere: *stânga dreapta*

- SAU EXCLUSIV pe biți `^`

Nivelul 12 de prioritate. Asociere: *stânga dreapta*

- SAU (SAU INCLUSIV) pe biți `|`

Nivelul 13 de prioritate. Asociere: *stânga dreapta*

- conjuncția logică `&&`

Nivelul 14 de prioritate. Asociere: *stânga dreapta*

- disjuncția logică `||`

Nivelul 15 de prioritate. Asociere: *dreapta stânga*

- atribuirea, atribuirea compusă `= += -= *= /= %= >>= <<= &= ^= |=`

- operatorul `,`

 Candale Silviu (silviu) • 22.11.2020

Du-te sus!