CAPITOLUL 5

5. Operatori ai limbajului C

Un operator reprezintă din punct de vedere al programării, acel simbol care alăturat la una sau două nume de variabile specifică operația ce trebuie realizată asupra lor. Funcție de numărul de variabile (deci de date) asupra căruia se aplică operatorii pot fi clasificați în două mari categorii:

- operatori unari, a căror folosire poate fi ca mai jos:
  - nume_var_i **oper_unar** nume_var_j;
  - **oper_unar** nume_var_i, unde i, j $\in$ N;

- operatori binari a căror folosire este ca mai jos:
  - nume_var_i=nume_var_j **oper_binar** nume_var_k, unde
    i, j, k $\in$ N.

Limbajul C are mai multe tipuri de operatori:

- operatori aritmetici care dau posibilitatea realizării oricăror calcule aritmetice;
- operatori relaționali ce sunt folosiți pentru testarea relațiilor ce pot exista între conținutul a două variabile;
- operatori logici folosiți pentru:
  - a putea realiza expresii complexe în cadrul testării relațiilor dintre variabile;
  - a procesa la nivel de bit variabilele.

5.1. Operatori aritmetici

Acest tip de operatori realizează calcule matematice simple. Desigur că precizia în care va face calculul respectiva aplicație ține de tipul cont de dată ales. Operatorii sunt prezentați în lista de mai jos. Să considerăm două variabile a şi b care la un moment dat conțin valorile a = 3 şi b = 5 atunci

- '+' adunare y=a+b→y=8
- '-' scădere y=a-b→y=-2
- '*' înmulțire y=a*b→y=15
- '/' împărțire y=a/b→y=0.6000000…0
- '%' modulo y=a%b→y=0 (restul împărțirii lui a la b)

5.2 Expresiile

O expresie constă dintr-un conglomerat realizat din:

- → operanzi
- → operatori

în scopul calculării unei valori.
Blocurile de bază constructive ale expresiilor pot fi :

- variabile
- constante

- apeluri funcție

Există patru mari tipuri importante de expresii:.

1. *Expresii constante* care reţin numai valori constante.

De exemplu 5
5+6*13/30
‘a’

2. *Expresii întregi* care sunt expresii care după realizarea tuturor conversiilor automate sau explicite de tip, produc un rezultat care este de tip între. Presupunând j,k întregi atunci următoarele expresii întregi:

   j
   j*k
   j/k+3
   k-‘a’
   3+(int)5.0

3. *Expresii în virgulă mobilă* care sunt expresii care după toate conversiile automate sau explivite de tip, produc un rezultat care este în virgulă mobilă. Dacă x este float sau double, atunci următoarele expresii sunt de acest tip:

   x
   x+3
   x/y*5
   3.0
   3.0-2
   3+(float) 4

4. *Expresii de tip pointer* care sunt expresii care în urma evaluării dau o adresă. De exemplu dacă p este pointer, iar j este întreg:

   p, p+1, (char*) 0x00ffffff, &j, “abc” sunt expresii de tip pointer.

5.2.1 Ordinea de evaluare a unei expresii funcție de precedența operatorilor.

Unul din primele lucruri de care trebuie ținut seama, când se scriu expresii aritmetice complexe, este faptul că spațiile albe (spațiu, tabulator, salt la linie nouă) nu au înțeles (nu sunt luate în considerare). De exemplu cele două expresii de mai jos sunt echivalente:

```
y=x*x+2*x-5*x/y;
y=x * x + 2 * x – 5 * x / y;
```

Oamenii folosesc câteodată spațiu în loc de paranteze pentru a specifica modul de evaluare a unei expresii. Noțiunea de evaluare se referă la ordinea în care se vor realiza calculele dintr-o expresie matematică. În cadrul unui limbaj de programare numai

parantezele joacă acest rol. În lipsa parantezelor dintr-o expresie se vor respecta regulile de precedență implicită.

Ca un exemplu să discutăm modul de evaluare a expresie de mai sus dacă se va ține seama numai de precedențele implicite. Deoarece operatorii {*, /, %} au aceeaşi precedență care însă este mai mare ca a operatorilor {+,-} evaluarea expresiei se va face de la stânga la dreapta după cum urmează:

1. se calculează valoarea lui x*x;
2. se calculează valoarea lui 2*x;
3. se calculează valoarea lui 5*x;
4. se calculează valoarea lui (5*x)/y;
5. se face suma (x*x)+(2*x);
6. se face scăderea (x*x)+(2*x) - (5*x)/y.

Noțiunea de precedență se referă la modul în care iau decizii în evaluarea unei expresii dacă nu am puse paranteze. Mai jos este prezentat tabelul cu toate precedențele implicite:

Operatori	Ordinea de precedență
() [] →	de la stânga la dreapta
! ~ ++ -- + - (unary) & (type) sizeof	de la dreapta la stânga
* / %	de la stânga la dreapta
+ -	de la stânga la dreapta
<< >>	de la stânga la dreapta
< <= > >=	de la stânga la dreapta
== !=	de la stânga la dreapta
&	de la stânga la dreapta
^	de la stânga la dreapta
|	de la stânga la dreapta
&&	de la stânga la dreapta
||	de la stânga la dreapta
?:	de la dreapta la stânga
= += -= *= /= %= &= ^= |= <<= >>=	de la dreapta la stânga
,	de la stânga la dreapta

5.3. Tipuri aritmetice de date

Numerele întregi sunt reprezentate diferit de cele în virgulă flotantă. De aceea apare următoarea problemă: conversia numerelor dintr – un format într-altul.

Dacă se doreşte ca rezultatul să fie stocat într-un alt format mai mult sau mai puțin cuprinzător atunci acesta trebuie depus într–o variabilă de respectivul tip şi conversia se va face automat. Există cazuri în care compilatorul, deşi acceptă conversia, dă un mesaj

de atenționare. Această situație apare atunci când se poate altera valoarea unor date deoarece formatul în care fac conversia este mai mic.

Dacă programatorul este sigur că doreşte respectiva conversie atunci trebuie să realizeze o conversie explicită pentru ca respectivul mesaj de atenționare să dispară. Deoarece la conversiile implicite se poate altera valoarea calculată poate apărea o eroare greu detectabilă.

S1= (int)((double)z/x+0.5);

În exemplul de mai sus se forțează, trunchierea de la o reprezentare de tip double la o reprezentare în numere întregi.

Datele nu pot fi alterate dacă se realizează o conversie de tip între un tip de dată cu domeniu de valori mai mic şi un tip de dată al cărui domeniu este mai mare.

5.4. Operatorii unari ++ şi - -

După cum s-a menționat aceşti operatori se numesc unari deoarece se aplică asupra unei singure variabile (expresii) la un moment dat. Ei realizează incrementarea sau decrementarea valorii respective cu o unitate.

Limbajul C mai are un avantaj: aceşti operatori pot fi folosiți pentru post sau preincrementare. Acest lucru este prezentat în exemplu de mai jos:
Dacă x=4 atunci:

Preincrementare	y=++x ←→ x=x+1=5	şi y=x=5
Predecrementare	y=--x ←→ x=x-1=3	şi y=x=3
Postincrementare	y=x++ ←→ y=x=4	şi x=x+1=5
Postdecrementare	y=x-- ←→ y=x=4	şi x=x-1=3

5.5. Operatorii op=

Au fot introduşi pentru simplificarea scrierii codului. O instrucțiune generică de tipul rez = rez op opd1 ←→ rez op = opd1. Aceşti operatori aparțin următoarei clase { += -= = /= %= &= ^= |=} de exemplu: x=x*7 ←→x*=7;

5.6. Operatori relaționali şi logici

Pentru a se putea lua decizii într-un program, deci pentru a realiza ramificarea fluxului de prelucrare a datelor, trebuie să existe o modalitate de comparare a informațiilor şi de asemeni de combinare a expresiilor logice primitiv în expresii mai complexe.

De exemplu dacă în variabilele x şi y am stocată valoarea 4 atunci expresia x==y va fi ADEVĂRATĂ şi va avea la evaluarea expresiei valoarea numerică 1, dacă x==4 şi y=5 atunci expresia x==y va fi FALSĂ şi deci în urma evaluării va fi înlocuită cu valoarea 0.

Operatori relaționali	Explicație
==	Egal cu
!=	Diferit de
>	Mai mare ca
<	Mai mic ca
>=	Mai mare sau egal
<=	Mai mic sau egal
Operatori logici	
||	SAU
&&	ŞI
!	NU

5.7. Expresii logice complexe

De obicei este insuficientă o singură expresie logică simplă. În scrierea unui program este necesară de multe ori compunerea a mai multe decizii. Un exemplu clasic este cazul în care trebuie verificată aparteneța valorii stocate într-o variabilă la un anumit interval. De exemplu $x \in [5,20]$ este echivalent cu $x \geq 5$ şi $x \leq 20$ ceea ce folosind operatori logici şi relaționali se poate scrie: (x>=5)&&(x<=20).

Tabelele de adevăr pentru operatorii logici ai limbajului sunt:

X	Y	X&&Y	X||Y	!X
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Urmează prezentarea un program simplu care realizează diverse operații logice cu un set de valori inițializate direct în program. Această metodă de inițializarea are dezavantajul că programul nu poate primi pentru calcul alte valori decât dacă se modifică codul sursă.

```
void main(void)
    {
    char mask;
    char number[6];// tablou de 6 elemente de tip char
    char and,or,xor,inv,index;
      number[0] = 0x00;// inițializare în cod
      number[1] = 0x11;
      number[2] = 0x22;
      number[3] = 0x44;
      number[4] = 0x88;
      number[5] = 0xFF;
      printf(" numerele asupra carora am aplicat diverse operatii logice folosind o
    masca sunt: \n");
```

```
    mask = 0x0F;
    for (index = 0;index <= 5;index++) {
      and = mask & number[index];// şi logic
      or = mask | number[index];//sau logic
      xor = mask ^ number[index];// sau exclusiv
      inv = ~number[index];// not bit cu bit
      printf("%5x %5x %5x %5x %5x %5x\n",number[index],
            mask,and,or,xor,inv);
    }

    printf("\n");
    mask = 0x22;
    for (index = 0;index <= 5;index++) {
      and = mask & number[index];
      or = mask | number[index];
      xor = mask ^ number[index];
      inv = ~number[index];
      printf("%5x %5x %5x %5x %5x %5x\n",number[index],
            mask,and,or,xor,inv);
    }
  }
```

Pentru restul tipurilor de operatori prezentați vom da un alt exemplu de folosire a lor.

```
void main(void)
  {
  int x = 0,y = 2,z = 1025;
  float a = 0.0,b = 3.14159,c = -37.234;
                                  /* incrementare */
    x = x + 1;      /*  x se incrementează */
    x++;           /* x se postincrementează */
    ++x;           /* x se preincrementează x */
    z = y++;        /* z = 2, y = 3 */
    z = ++y;        /* z = 4, y = 4 */

                                  /* decrementare */
    y = y - 1;      /* y se decrementează */
    y--;           /* y este postdecrementat */
    --y;           /* y este predecrementat */
    y = 3;
    z = y--;        /* z = 3, y = 2 */
    z = --y;        /* z = 1, y = 1 */

                                  /* operații aritmetice */
    a = a + 12;     /* se adună 12 la a */
    a += 12;        /* se adună încă 12 la a */
```

```
  a *= 3.2;       /* a se înmulţeşte cu 3.2 */
  a -= b;         /* se scade b din a */
  a /= 10.0;      /* se împarte a la 10.0 */

                          /* expresii condiţionale */
  a = (b >= 3.0 ? 2.0 : 10.5 );     /* această expresie    */

  if (b >= 3.0)                /* este identică cu toată instrucţiunea if… */
    a = 2.0;                      /* din punct de vedere al rezultatului*/
else
    a = 10.5;

  c = (a > b?a:b);       /* c primeşte maximum dintre a şi b */
  c = (a > b?b:a);       /* c primeşte minimul dintre a şi b */
}
```