3. OPERATORI

Scopul oricărui program de calcul este de a prelucra datele stocate în variabile, în conformitate cu un algoritm de calcul, pentru a obține rezultate. Variabilele, deşi constituie o parte importantă a unui limbaj de programare, fără un set de operatori şi un set de instrcțiuni care să stabilească ce operații se efectuează asupra lor şi cum se fac aceste operații, nu au nici o importanță.

În limbajul C există o multitudine de operatori, care pot fi grupați în următoarele clase principale:

- operatori aritmetici;
- operatorul de atribuire şi de atribuire aritmetică;
- operatori de incrementare şi de decrementare;
- operatori relaționali;
- operatori logici;
- operatori la nivel de bit.

3.1. Operatori aritmetici

Limbajul C foloseşte 5 operatori aritmetici : **+** (adunarea a două valori); **-** (scăderea a două valori); ***** (înmulțirea a două valori; **/** (împărțirea a două valori); **%** (restul împărțirii a două valori). Efectul primelor 4 operatori este bine cunoscut din algebră. Operatorul *%* reprezintă operatorul algebric **modulo n** şi furnizează restul împărțirii unui număr întreg la n. De exemplu, o instrucțiune de forma: *a=13%5;* va atribui variabilei de tip întreg *a* valoarea *3*.

Considerăm programul de calcul din exemplul 3.1, care realizează transferarea în grade Celsius şi Kelvin a unei temperaturi exprimate în grade Fahrenheit:

Exemplu 3.1

```
/* Programul ex_3_1*/
#include <stdio.h>
void main (void)
 {
    float ftemp, ctemp, ktemp=273.15;
    printf("\n Introduceti temperatura in grade Fahrenheit >");     scanf("%e",&ftemp);
    ctemp = (ftemp-32)*5/9;              ktemp+= (ftemp-32)*5/9;
    printf("\n Temperatura in grade Celsius %g. Temperatura in grade Kelvin %g",ctemp,ktemp);
 }
```

Referitor la modul în care limbajul C avaluează expresiile în care intervin operatorii aritmetici, sunt necesare următoarele precizări: 1) ca şi în algebră, în expresiile, în care nu intervin paranteze, mai întîi se efectuează operațiile de înmulțire şi împărțire şi apoi de adunare şi scădere. Spunem, că înmulțirea şi împărțirea **preced** adunarea şi scăderea, adică operatorii * şi / au o prioritate mai mare decît operatorii + şi -; 2) în expresiile, în care intervin paranteze, acestea vor fi efectuate începînd cu perechea cea mai interioară şi terminînd cu perechea superioară.

3.2. Operatori de atribuire şi de atribuire aritmetică

În limbajul C se folosesc o serie de operatori, care comprimă instrucțiunile. Considerăm următoarea instrucțiune de atribuire: *a = a+b;* în care valoarea lui *b* este adunată cu valoarea lui *a* şi rezultatul este atribuit variabi-lei *a* prin intermediul operatorului de atribuire "=".

În limbajul C această instrucțiune poate fi scrisă astfel *a+=b;*. Efectul este acelaşi dar expresia este mai compactă. Toți operatorii aritmetici pot fi combinați cu semnul egal în acelaşi mod, obținîndu-se **operatorii de atribuire aritmetică** prezentați în tabelul 5.

Tabelul 5

Operator	Scop	Exemple	Forma echivalentă în Pascal
=	atribuire simplă	a = 15	a:=15
+=	adunare cu atribuire	a += b	a := a+b
-=	scădere cu atribuire	a -= b	a := a-b
*=	înmulțire cu atribuire	a *= b	a := a*b
/=	împărțire cu atribuire	a /= b	a := a/b
%=	atribuire modulo n	a %= b	a := a mod b

O altă modalitate de a compacta codul o constituie utilizarea expresiilor în locul variabilelor. Acest lucru se poate constata şi din exemplul următor în care s-a rescris într-o formă compactă, programul de conversie a temperaturilor.

Exemplul 3.2.

```
/* programul Ex_3_2 */
# include <stdio.h>
void main (void)
 {
    float ftemp, ktemp=273.15;
    printf("Introduceți temperatura în grade Fahrengheit >");     scanf("%e", &ftemp);     ktemp+-=(ftemp-32)*5/9;
    printf("\n Temperatura în grade Celsius %g temperatura în grade Kelvin %g ", (ftemp-32)*5/9, ktemp);
 }
```

3.3. Operatorii de incrementare şi decrementare

Operatorul de incrementare, simbolizat ++ şi **operatorul de decrementare**, simbolizat — sunt specifici limbajului C şi au ca efect creşterea cu 1, respectiv micşorarea cu 1, a valorii variabilei, căreia i se aplică. Considerăm următorul program:

Exemplul 3.3.

```
/* Programul ex_3_3 */
# include <stdio.h>
void main (void)
```

```
{
    int i=0, j=0;
    printf("\ni=%4d\tj=%4d",i,j);
    printf("\ni=%4d\tj=%4d",i++,--j);
    printf("\ni=%4d\tj=%4d",i,j);
}
```

Dacă rulăm acest program vom obține pe ecran următorul rezultat:

```
i=0j=0
i=0j=-1
i=1j=-1
```

Primul apel al funcției **printf** afișează valorile inițiale ale lui *i* şi *j* care sunt egale cu 0, ca efect al primei instrucțiuni din program. În al doilea apel al funcției **printf** sunt folosiți operatorii de incrementare a valorii lui *i* şi respectiv *j*. Din rezultatul obținut observăm că valoarea lui *j* s-a modificat devenind -1. Acest rezultat explică modul în care acționează operatorii ++ şi --, sintetizat în următoarele:

- dacă opeeratorul ++ sau -- este folosit în fața numelui variabilei, atunci mai întîi se modifică valoarea variabilei şi apoi este folosită noua valoare. Acest mod de utilizare poartă numele de **antidecrementare**, respectiv **antiincrementare**.
- dacă operatorul ++ sau -- este folosit după numele variabilei, atunci mai întîi este folosită vechea valoare, după care valoarea este modificată. Acest mod de utilzare se numeşte **postincrementare**, respectiv **postdecrementare**.

Operatorii ++ şi -- pot deci incrementa sau decrementa o variabilă, fără a fi necesară o instrucțiune separată, contribuind şi ei la compactarea programului.

3.4. Operatorii relaționali

Operatorii relaționali sunt simboluri cu ajutorul cărora construim întrebările privind relațiile existente între variabile. Acestea se numesc **expresii relaționale** şi pot fi *adevărate* sau *false*.

Considerăm pentru exemplificare, un proogram de calcul care citeşte de la tastieră vîrsta unei persoane, pe care o compară cu vîrsta majoratului, adică cu valoarea 18.

Exemplul 3.4.

```
/* programul ex_3_4 */
# include <stdio.h>
void main (void)
 {
    int virsta;
    printf("\n Introduceți virsta >");     scanf("%d",&virsta);
    printf("\n Este virsta mai mica decit 18? %d", virsta<18);
 }
```

În limbajul C o expresie relațională are valoarea 1 dacă expresia este adevărată şi valoarea 0, dacă expresia este falsă. Deci, spre deosebire de Pascal, în care adevărat şi fals sunt reprezentate prin variabile speciale de tipul "boolean", în limbajul C acestea sunt reprezentate prin valorile întregi 1 (sau orice altă valoare întreagă diferită de 0) şi, respectiv, 0. Expresiile relaționale constituie cheia instrucțiunulor de ciclare şi selecție.

Limbajul C foloseşte şase operatori relaționali: 1) < strict mai mic; 2) <= mai mic sau egal; 3) > strict mai mare; 4) >= mai mare sau egal; 5) == egal cu; 6) **!=** diferit de.

Într-o expresie relațională pot fi folosiți, pe lîngă operatorii relaționari, şi operatorii aritmetici, ca în exemplul următor:

Exemplul 3.5.

```
/* programul ex_3_5 */
# include <stdio.h>
void main (void)
 {
    int val;
    printf("\n Introduceți o valoare întreagă >");scanf("%d",&val);
    printf("\n Este %d egal cu 5+7 ? %d",val,val==5+7);
 }
```

Dacă se rulează acest program şi se tastează numărul 12 la solicitarea valorii întregi, atunci pe ecran apare mesajul: *Este 12 egal cu 5+7 ? 1*, iar d acă se tastează orice altă valoare, de exemplu, numărul 11, atunci pe ecran apare mesajul: *Este 11 egal cu 5+7 ? 0*. Răspunsurile obținute sunt corecte şi se explică prin faptul că operatorii relaționale au o prioritate mai mică decît operatorii aritmetici.

Din cele precizate se poate constata, că operatorul relațional "egal cu", simbolizat prin == este total diferit de operatorul de atribuire "egal", simbolizat prin =. Astfel, expresia *val == 5+7* care compară valoarea variabilei *val* cu expresia aritmetică *5+7*, răspunsul fiind 0 sau 1, este total diferită de expresia *val = 5+7* care atribuie variabilei *val* valoarea *5+7*, adică 12.

3.5. Operatori logici

În cadrul programelor de calcul pentru luarea unei decizii cîteodată este necesar să combinăm mai multe expresii relaționale, adică să efectuăm operații cu valorile *adevăr* şi *fals*, asociate cu valorile întregi *1* şi respectiv *0*. Aceste operații sunt operații din logica matematică: **sau logic** (**OR**), **şi logic** (**AND**), respectiv **negația logică** (**NOT**). Aceşti trei operatori logici sunt utilizați în limbajul C, împreună cu operatorii relaționali, pentru construcția expresiilor logice necesare în luarea deciziilor. Simbolizarea acestor operatori logici este următoarea:|| "sau logic" (OR), **&&** "şi logic" (AND), ! "negația logică" (NOT).

Considerăm următorul program, care solicită apăsarea unei taste şi răspunde la întebarea dacă tasta apăsată este o cifră sau un alt caracter.

Exemplul 3.6

```
/* programul ex_3_6 */
# include <stdio.h>
void main (void)
 {
    char ch;
    printf("\n Apăsați o tastă: "); ch=getche( );
    if (ch >=48 && ch <=57) printf ("este o cifră");
    else                    printf ("nu este o cifră");
 }
```

Cheia acestui program o constitue expresia logică: *ch >=48 && ch <=57* construită cu operatorii raționali >= și <= și operatoril logic **&&**,care arată,că dacă ambele expresii relaționale sunt adevărate,atunci întreaga expresie este adevărată. Aceasta se întîmplă doar dacă valoarea vareabilei *ch* este cuprinsă între 48 și 57, adică reprezintă codul ASCII al unei cifre.

În limbajul C operatorii logici au o prioritate mai mică decît operatorii relaționali.

3.6. Operatorii la nivel de bit

Așa cum se știe datele sunt reprezentate în memoria internă în formă binară. Pînă în prezent am utulizat tipurile de date (caracter, întreg, e.t.c.) cu care am efectuat operații fără a vedea cum sunt combinate la nivelul biților și cum pot fi aceștea accesați în mod individual. Accesarea individuală a biților este necesară atunci, cînd în cadrul unui program de calcul pur gramatical programatorul interacționează direct cu mașina.

Una din trăsăturile fundamentale ale limbajului C o constitue posibilitatea de a manevra cu biții individuali cu ajutorul următoarelor operații: **&** - "și logic" la nivel de bit (AND); **|** - "sau logic" la nivel de bit (OR); **^** - "sau exclusiv" la nivel de bit (XOR); **>>** - transfer la dreapta a biților; **<<** - transfer la stînga a biților; **~** - "complementul" la nivel de bit.

Acești operatori pot fi aplicați caracterelor și întregilor cu sau fără semn, dar nu pot fi aplicați numerelor reale. Deoarece operatorii la nivel de bit acționează asupra biților individuali, este important să precizăm că aceștea sunt numerotați de la dreapta la stînga (figura 1).

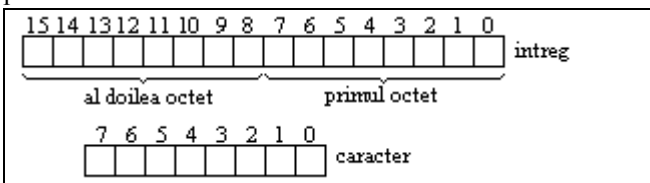

Figura 1.

Pentru a explica modul în care lucrează operatorii la nivel de bit, considerăm 2 variabile de tip caracter, a și b, cărora le atribuim valorile hexazecimale F0 și 0F: a = 1111000; b = 00001111.

Operatorul & (AND) se aplică la doi operanzi de același tip realizînd "**și logic**" bit cu bit:

```
          11110000
&         00001111
          00000000
```

În mod obișnuit, operatorul & este utilizat pentru a testa dacă un bit este setat pe 0 sau 1. Procesul de testare constă în efectuarea unei operații AND la nivel de bit între variabila, al cărea conținut îl testăm și o constantă hexazecimală, numită **mască**, care are setat pe 1 doar bitul ce urmează a fi testat.

De exemplu, pentru a testa dacă bitul 4 al variabilei de tip caracter *car* este setat pe 1 sau 0 vom folosi următoarea instrucțiune *bit4 = car & 0x10;*.

Să exemplificăm procesul de testare pentru două valori ale variabilei *car*.

```
                                  01011011
1) car & masca = 0x5B & 0x10 = 0x10   &   00010000
                                  00010000
                                  01101011
2) car & masca 0x6B & 0x10 = 0x00 &   00010000
                                  00000000
```

Din exemplele prezentate se constată că rezultatul operației este diferit de 0, în cazul în care bitul respectiv este setat pe 1 și 0, în cazul în care este setat pe 0.

Operatorul | (OR) se aplică la 2 opranzi de același tip realizînd "**sau logic**" bit cu bit:

```
          11110000
|         00001111
          11111111
```

În mod obișnuit operatorul **|** este utilizat pentru a combina biții din două variabile într-una singură. Considerăm, de exemplu, două variabile *car1* și *car2* de tipul caracter. Presupunem, că biții de la 0 la 3 ai lui *car1* conțin o valoare, iar biții de la 4 la 7 a lui *car2* conțin o altă valoare și că dorim să punem împreună cele două valori în aceiași variabilă *car* de tipul caracter. Acest lucru se realizează prin folosirea instrucțiunii *car = car1|car2;*

```
                                  00000101
Exemplu : car1| car2=0x05| 0xB0=0xB5   |   10110000
                                  10110101
```

Operatorul ^ (XOR) se aplică la 2 operanzi de același tip realizînd operația logică "**sau exclusiv"** bit cu bit conform următoarei reguli: 0^0 = 1^1= 0; 0^1= 1^0 = 1. Acest operator este utilizat fie pentru a șterge un bit, fie pentru a-l seta. Astfel, dacă dorim ștergerea bitului 7 și setarea bitului 3 al variabilei *a*, vom folosi instrucțiunea *a = a ^ 0x88;* Fie *a = 0xF0*. Ca rezultat avem a = 0x78.

Se constată cu ușurință, că dacă se repetă operația *a = a ^ 0x88* se va seta bitul 7 și se va șterge bitul 3, asfel ca variabila *a* recapătă valoarea avută inițial.

Operatorii >> și **<<** se aplică unui singur operand de tipul întreg sau caracter. Ei au ca efect translatarea la dreapta, respectiv la stînga a biților cu un număr de poziții egal cu numărul ce urmează operatorului.

În cazul operatorului >>, pozițiile din stînga rămase libere vor fi completate cu 0 sau 1, în funcție de tipul și valoarea variabilei asupra căreia se operează. În cazul variabilelor de tip caracter și întreg, bitul cel mai din stînga este folosit pentru semn (1 pentru

semnul “-”, 0 pentru semnul “+”), iar restul biților pentru memorarea valorii efective. În cazul variabilei de tipul caracter şi întreg declarate fără semn (**unsigned**) se folosesc toți biții pentru memorarea valorii. Deci, la aplicarea operatorului >> asupra unei variabile ce conține o valoare negativă, pentru păstrarea semnului “-”, pozițiile din stînga rămase libere vor fi completate cu 1. În cazul variabilelor fără semn sau cu valori pozitive operatorul >> va completa cu 0 pozițiile rămase libere în stînga. Fie a=0xF0= = 11110000 - variabilă declarată de tipul caracter fără semn, atunci valoarea lui *a* după evaluarea expresiei *a=a>>1;* - 11111000.

Aşa cum se poate constata, translatarea la dreapta cu o poziție este echivalentă cu o împărțire prin 2.

În cazul **operatorului <<** pozițiile din dreapta rămase libere vor fi complectate cu 0, indeferent de tipul şi valoarea operatorului. Astfel dacă b = 0x0F = 0001111, atunci valoarea lui *b* după evaluarea expresiei b=b>>1: 00011110.

Se constată că transltarea în stînga cu o poziție este echivalentă cu o înmulțire cu 2 a valorii operandului.

Operatorul ~ se aplică unui singur operant şi efectuiază o operație de negare logică la nivel de bit. Deci el va transforma toți biții setați 1 în 0, iar pe cei setați 0 în 1. Astfel dacă a=0xF0=11110000, atunci ~a=00001111=0x0F. Pentru a exemplifica cum sunt utilizați operatorii la nivel de bit, în exemplul 3.7 este prezentat un program care realizează transformarea unei vvariabile de tip întreg din hexazecimal în binar.

Exemplul 3.7

```
/* Programul ex_3_7 */
# include (stdio.h)
void main (void)
 {
    int i, numar, bit;
    unsigned masca=0x8000;
    printf (“\nIntroduceti un numar hexazecimal >“);      scanf(“%x”,&numar);
    printf(“\nforma binara a lui %x este :”,numar);
    for (i=0; i<=15; i++)
     {
            bit(masca & numar) ?1 : 0;
            printf(“%2d”,bit);
            if(i==7) printf(“--”);
            masca = masca>>1;
     }
 }
```

Programul solicită introducerea unui întreg hexazecimal pe care va afişa în formă binară. Pentru aceasta în cadrul unei bucle se extrage cu ajutorul operatoruluii & şi apoi se tipăreşte valoarea lui. Cheia acestui program o constituie instrucțiunea *bit=(masca & număr) ? 1:0;* în care se foloseşte semnul “?” cu operator de întrebare. Aceasta instrucțiune este o exemplificarea a unei instrucțiuni de forma: *var = (expresie logică) ? val1:val2;* care funcționează astfel: dacă expresia logică este adevărată, variabila var va primi valoarea var1, iar dacă este falsă, atunci var va primi valoarea var2.

Revenind la exemplul prezentat, variabila *bit* va primi valoarea *1* dacă expresia *masca & număr* este adevărată şi 0, dacă acesta este falsă.

Deoarece valoarea inițială a variabilei masca este 0x8000, se extrage bitul 15, apoi masca este translatată la dreapta cu o poziție, căpătînd valoarea 0x4000, care va permite extragerea bitului 14. Acest proces continuă pînă cînd sunt extraşi şi afişați toți biții valorii introduse. Pentru a demarca cei 2 octeți ai variabilei număr du extragerea bitului 8 programul tipăreşte două caractere --.