Laborator 6 – Operatori

În marea lor majoritate, acţiunile desfăşurate în cadrul oricărui program se datorează expresiilor.

Expresiile reprezintă o combinaţie validă între operatori şi operanzi. **Operanzii** sunt de mai multe feluri şi sunt caracterizaţi de valoare şi tip. **Operatorii** sunt simboluri utilizate pentru a preciza ce operaţii trebuie executate asupra unor operanzi.
Operanzii pot fi de mai multe tipuri:

- constante de orice tip;
- numele unei variabile simple;
- numele unui tablou;
- numele unei structuri;
- numele unui tip;
- numele unei funcţii;
- numele unui element de tablou (variabilă cu indici);
- elementul unei structuri;
- apelul unei funcţii;
- expresie inclusă între paranteze rotunde.

Operatorii de incrementare-decrementare permit incrementarea şi decrementarea (mărirea/micşorarea cu o unitate) a valorii unei variabile. Sunt operatori unari si au aceeaşi prioritate cu ceilalţi operatori unari (tabelul 1).

Tab. 1.

Operatori	Semnificaţie	Tipul datelor	Exemple
Prefixaţi			
++operand	pre-incrementare	numeric	++k
--operand	pre-decrementare	numeric	--k
Postfixaţi			
operand++	post-incrementare	numeric	k++
operand--	post-decrementare	numeric	k--

În tabelul 2 sunt prezentaţi operatorii relaţionali şi logici ai limbajului C.

Tab. 2

Operatori	Semnificaţie	Exemple
relaţionali		
<	mai mic	a<13
<=	mai mic sau egal	a<=56
>	mai mare	a>90
>=	mai mare sau egal	f>=45
= =	egal	caract = ='\n'
!=	diferit	caract !='\n'
logici		
&&	ŞI logic	x&&y
| |	SAU logic	x| |z
!	NEGARE logică	!x

Ideea de "adevărat" sau "fals" stă la baza conceptelor de operatori relaţionali şi logici. În limbajul C „adevărat" este sinonim cu o valoare diferită de 0, iar „fals" cu 0. Operanzii din expresiile logice pot fi orice valori numerice chiar şi adrese.

Limbajul C acceptă o serie de **operatori pe biţi**, spre deosebire de alte limbaje. Cu ajutorul acestor operatori se pot efectua operaţii la nivel de bit. Operaţiile la nivel de biţi includ testarea, amplasarea sau deplasarea biţilor dintr-un cuvânt (2 octeţi). Operatorii pe biţi se aplică doar operanzilor de tip întreg.
În tabelul 3 este prezentată lista operatorilor pe biţi.

Tab 3.

Operator pe biţi	Semnificaţie	Exemple
&	ŞI logic pe biţi	n&234
|	SAU logic pe biţi	j | 56
^	SAU EXCLUSIV logic pe biţi	d ^ 32
~	NU complement faţă de 1	~g
<<	deplasare stânga	j<<2
>>	deplasare dreapta	d>>3

Primii 4 operatori se numesc operatori logici pe biţi. Ultimii doi operatori se numesc operatori de deplasare.
Operatorii ŞI, SAU şi NU sunt guvernaţi de aceeaşi tabelă de adevăr ca şi operatorii echivalenţi logici, dar lucrează la nivel de biţi.
Aşa cum reiese din tabel este "adevărat" numai dacă exact unul din operanzi este adevărat. În caz contrar este fals.

Operaţiile la nivel de bit se utilizează în programele de interfaţă cu echipamentele, cum ar fi programele de modem, rutine de fişiere de pe disc sau imprimantă. Operaţiile la nivel de biţi pot masca anumiţi biţi cum ar fi cei de paritate (bitul de paritate este de obicei bitul de ordinul cel mai mare din fiecare octet).

Operatorii de deplasare a biţilor >> şi << deplasează toţi biţii dintr-o variabilă la dreapta sau la stânga. Operaţiile de deplasare pot servi la decodificarea datelor de intrare provenite de la un dispozitiv extern (de exemplu un convertor digital-analog) precum şi la citirea informaţiei de stare. Cu ajutorul operatorilor de deplasare se pot de asemenea înmulţi sau împărţi foarte rapid numerele întregi. Operaţia de deplasare spre stânga este echivalentă cu înmulţirea cu puteri ale lui 2, iar operaţia de deplasare la dreapta este echivalentă cu o împărţire cu puteri ale lui 2 (tabelul 4).

Tab. 4.

int a;	a în timpul execuţiei declaraţiei	Valoarea lui a
a=7;	0000000000000111	7
a=a<<1;	0000000000001110	14
a=a<<3;	0000000001110000	112
a=a<<2;	0000000111000000	448
a=a>>1;	0000000011100000	224
a=a>>2;	0000000000111000	56

Operatorul de atribuire în limbajul C este "=". În ceea ce priveşte prioritatea operatorului este mai mică decât toţi operatorii întâlniţi până acum.
El se utilizează în expresii de forma:

nume_variabila=expresie;

În urma acestei operaţii variabilei nume_variabila i se atribuie valoarea expresie.
Principala problemă pe care o ridică operaţia de atribuire este cea a conversiei dacă acest lucru este necesar. Astfel valoarea expresiei din dreapta semnului de atribuire se va converti spre tipul variabilei din stânga acestuia, înainte de a se face atribuirea, dacă cele două tipuri sunt diferite.
În tabelul 5 se găsesc principalele reguli de conversii de tip.

Tab. 5.

Tipul membrului stâng al atribuirii (destinaţia)	Tipul expresiei	Pierderi posibile de informaţie
signed char	char	dacă valoarea >127, destinaţia<0
char	short int	cei mai semnificativi 8 biţi
char	int	cei mai semnificativi 8 biţi
char	long int	cei mai semnificativi 24 biţi
int	long int	cei semnificativi 16 biţi
int	float	partea zecimală şi posibil mai mult
float	double	precizie, rezultat rotunjit
double	long double	precizie, rezultat rotunjit

Observaţie:
Pentru a realiza o conversie care nu a fost prezentată în tabelul 5. se va proceda la conversii succesive. Pentru a converti o variabilă de tip double la una de tip int, mai întâi se face conversia de la double la float şi apoi de la float la int.

Limbajul C pune la dispoziţie alături de semnul "=" (declaraţia de atribuire) şi alţi operatori de atribuire combinată dată de succesiunea de caractere (tabelul 6):

operator=

unde operator reprezintă un operator aritmetic binar sau logic pe biţi.

Această formă alternativă a declaraţiei de atribuire este cunoscută si sub numele de forma scurtă C (C shorthand) care simplifică codificarea unor operaţii de atribuire.

nume_variabila operator=expresie ⇔ nume_variabila=nume_variabila operator expresie

Tab. 6.

Operatori de atribuire combinată	Forma lungă	Forma scurtă
+=	a=a+b;	a+=b;
-=	a=a-b;	a-=b;
*=	a=a*b;	a*=b;
/=	a=a/b;	a/=b;
%=	a=a%b;	a%=b;
&=	a=a&b;	a&=b;
|=	a=a|b;	a|=b;
^=	a=a^b;	a^=b;
<<=	a=a<<b;	a<<=b;
>>=	a=a>>b;	a>>=b;

Operatorul active la compilare sizeof este un operator unar activ la compilare care calculează lungimea în octeţi a variabilei sau a specificatorului de tip, încadrat între paranteze pe care îl precede. În ceea ce priveşte prioritatea operatorului sizeof, aceasta este aceeaşi cu cea a operatorilor unari ai limbajului C.

Forma sa generală este:

1. sizeof data sau sizeof (data)

sau

2. sizeof (tip date)

data poate fi:

nume de variabilă simplă⇒rezultatul va fi numărul de octeţi alocaţi variabilei;

nume de tablou ⇒ rezultatul va fi numărul de octeţi al zonei de memorie alocate tabloului;

nume de structură ⇒ rezultatul va fi numărul de octeţi al zonei de memorie alocate structurii;

referirea la elementele unui tablou(variabilă cu indici)⇒rezultatul va fi numărul de octeţi alocaţi elementului respectiv;

referirea la elementele unei structuri⇒rezultatul va fi numărul de octeţi alocaţi elementului respectiv.

tip date poate fi:

un cuvânt cheie al unui tip predefinit de date;

o construcţie care defineşte un tip.

Observaţie:

Pentru a calcula dimensiunea unui tip de date numele tipului trebuie inclus între paranteze. Acest lucru nu este neapărat necesar în cazul variabilelor.

Operatorul cast are rolul de a forţa o expresie să fie de un anumit tip. El se numeşte şi operator de conversie explicită.

Forma generală a operatorului cast este:

(tip) expresie

unde tip este un tip de date recunoscut de C.

Operatorul cast este un operator unar şi se bucură de aceeaşi prioritate ca alţi operatori unari.

Exemplu:

```
int i;
(float)i/2;
```

Expresia i/2 este evaluată de tip float şi se asigură afişarea părţii fracţionare. În lipsa lui cast (float) ar fi fost efectuată numai o împărţire întreagă.

Observaţii:
1. Conversia forţată de la long la int, short şi char duce la pierderea biţilor semnificativi din stânga.
2. Conversia forţată de la float la int duce la trunchierea părţii fracţionare sau eventual şi a părţii întregi, deoarece tipul float are un interval mai mare decât int.
3. Conversia de la int la unsigned int nu duce la nici o modificare, doar informaţia din zona respectivă se interpretează altfel.

Operatorul adresă "&". Este un operator unar şi are aceeaşi prioritate ca şi alţi operatori unari.

Forma sa generală este:
&nume
nume reprezintă numele unei variabile simple sau al unei structuri.
Exemplu.
a=&cont;

Efect: plasează în a adresa de memorie a variabilei cont. Această adresă reprezintă localizarea internă a variabilei în interiorul calculatorului şi nu are nici o legătură cu valoarea variabilei cont. Dacă presupunem că variabila cont se află la locaţia de memorie 1000, iar valoarea lui cont=300. Conform declaraţiei precedente, a primeşte valoarea 1000.
Deci declaraţia de mai sus are următorul sens:
"a primeşte adresa variabilei cont".

Dacă nume este un nume de tablou, nu se mai utilizează operatorul adresa &, fiindcă nume are ca valoare chiar adresa de început a zonei de memorie alocate tabloului.

Operatorul adresă "*" este complementul lui "&".
Este un operator unar şi are aceeaşi prioritate ca şi ceilalţi operatori unari ai limbajului.
El calculează valoarea variabilei localizate la adresa care urmează.

Exemplu.
p=*a;
Efect: amplasează valoarea lui cont în p. Deci p are valoarea 300, fiindcă 300 este stocat la locaţia 1000, adresa de memorie la care se găsea a.
Deci declaraţia de mai sus are semnificaţia: "p primeşte valoarea situată la adresa a".

Limbajul C oferă un operator foarte eficient şi comod de folosit "?:", care înlocuieşte declaraţiile de forma if-then-else (vezi capitolul 6). Este un operator ternar.

Operatorul condiţional "?:" apare în expresii condiţionale cu forma generală:
exp1 ? exp2 : exp3;
Efectul acestui operator este următorul:
Se evaluează exp1.
Dacă exp1 este adevărată (diferită de 0) , se evaluează exp2 iar tipul şi valoarea sa reprezintă tipul şi valoarea expresiei condiţionale;
Dacă exp1 este falsă (egală cu 0), se evaluează exp3 iar tipul şi valoarea sa reprezintă tipul şi valoarea expresiei condiţionale.
În ceea ce priveşte prioritatea operaţiilor operatorul condiţional are prioritatea imediat mai mică decât prioritatea operatorului SAU logic "||" şi imediat mai mare decât prioritatea operatorului de atribuire. El se asociază de la dreapta la stânga.
În tabelul 7 este prezentată ierarhia tuturor operatorilor limbajului C. Valorile din partea dreaptă a tabelului cuantifică precedenţa diverşilor operatori de la cei cu prioritate maximă (precedenţa 1) până la cei cu prioritate minimă (precedenţa 15). Toţi operatorii limbajului C se asociază de la stânga la dreapta. Excepţie fac operatorii unari şi a operatorului ?= care se asociază de la dreapta la stânga.

Tab. 7

Operatori	Precedenţa
() [] -> .	1
! ~ ++ - (tip) * & sizeof	2
* / %	3
+ -	4

<< >>	5
< <= > >=	6
== !=	7
&	8
^	9
|	10
&&	11
||	12
?:	13
= += -= *= /= etc.	14
,	15

Exemple

1. Rulaţi programul de mai jos şi urmăriţi ce se întâmplă în fereastra watch:

```
void main(void)
{
int i=20000, j=15000,k;
float r;
printf(”Introduceti valoarea lui k=”);
scanf(”%d”,&k);
printf(”\n%d %p\n”,k,&k);
k=i+j; r=i+j;
r=(float)i+j; r=(long)i+j;
r=i/j; k=i/j; r=(float)i/j;
k=i%j;
k=+ - - -i; k=+ - --i; k=- +j--; i=k/(j-j);
}
```

2. Scrieţi un program care evaluează expresia de mai jos, ţinând seama că a=1, x=2.03, y=5.11.

e=!(x= =y| |a)&&x>y**2+1| |x!=y-1

3.

```
int a;
char caract;
float ge;

void f(void)
{
caract=a;
a=ge;
ge=caract;
ge=a;
}
```

În funcţia f( ), atribuirile au următoarele semnificaţii:
Prima atribuire: biţii de ordinul cel mai mare al variabilei a de tip int sunt înlăturaţi lăsându-se variabilei caract numai cei 8 biţi inferiori (1 octet).
A doua atribuire: lui a i se atribuie partea nefracţionară a lui ge.
A treia atribuire: ge convertește valoarea întreagă pe 8 biţi stocată în variabila caract în aceeaşi valoare dar în format virgulă mobilă.
A patra atribuire: variabila ge convertește o valoare întreagă în format virgulă mobilă.

4. Utilizarea operatorului adresă:

```
#include <stdio.h>
void main(void)
{
int d, s;
```

```
int *m;
s=20;
m=&s;
d=*m;
printf("Valoarea lui d=%d\n", d);
}
```

5. Utilizarea operatorului sizeof:

```
#include <stdio.h>
void main (void)
{
float p1;
printf("Dimensiunea datei p1:%d\n", sizeof p1);
printf("Dimensiunea tipului de data int:%d\n", sizeof(int));
}
```

6. Rulaţi programul de mai jos şi urmăriţi ce se întâmplă în fereastra watch:

```
void main(void)
{
int a,b,c;
float z;
a=25000; b=20000;
c=a+b; z=a+b;
c=(float)a+b; z=(float)a+b;
c=a/b; c=a%b; c=a>b; c=a==b;
a=3; b=11;
c=a++ + ++b; c=a&b; c=a|b;
c=b<<2; c=-a>>3; c=~a;
a=0; c=a&&b; c=a||b; c=!a;
a=4;c*=a;
c=(a>b)?a:b
}
```

7. Ilustrarea conversiilor:

```
#include <stadio.h>
#include <conio.h>
void main(void)
char c=’a’, cc;
int i=4;
float f=5.95;
printf(”%d %f\n”, i, f);
i=f; //conversie implicită, trunchiere
printf(”%d %f\n”, i, f);
f=i+100000; //conversie implicită a rezultatului expresiei
printf(”%d %f\n”, i, f);
i=-99.001;
f=’a’;
}
```

8. Scrieţi un program care evaluează expresia de mai jos, ţinând seama că A=-3, B=4.3, C=0, D=-3.

E = ((A||B&&D)&&C) && !(A==-D)

Şi aceeaşi expresie pentru C =4;
Care este diferenţa şi de ce?