Laboratorul 4. OPERATORI ŞI EXPRESII

Introducere

Orice program transpune o problemă prin intermediul unei succesiuni de instrucțiuni, fiecare producând o acțiune. Aceste instrucțiuni includ în mod frecvent expresii formate din succesiuni de operanți şi operatori.

Operatorii sunt simboluri care, aplicate unor operanzi (date - constante, variabile, expresii), au ca efect realizarea unor operații care returnează un rezultat.

O succesiune validă de operanzi şi operatori formează o **expresie.**

Limbajele C/C++ dispun de o gamă largă de operatori. C++ posedă toți operanzii limbajului C, completând lista cu operanzi proprii.

Operanzii pot fi grupați după diferite criterii, de exemplu după numărul operanzilor cărora li se aplică sau după tipul operației efectuată.

4.1. Operatori

Operatorii sunt simboluri care aplicate unor **operanzi** (date - constante, variabile, expresii) au ca efect realizarea unor operații care returnează un rezultat.

O succesiune validă de operanzi şi operatori formează o **expresie**.

Limbajele C/C++ dispun de o gamă largă de operatori.

C++ posedă toți operanzii limbajului C, completând lista cu operanzi proprii: operatorii **new** şi **delete** utilizați pentru alocarea dinamică de memorie, operatorul de scop (::), operatorul pointer la membru (.*) şi forma sa echivalentă (->.) .

Operatorii pot fi grupați după diferite criterii. Unul dinte criterii îl constituie tipul operației. Astfel, operatorii pot fi grupați în operatori **aritmetici**, operatori **relaționali**, operatori **la nivel de bit** şi **operatori logici**. În afară de aceste categorii, există o serie de operatori care descriu operații diverse.

Alt criteriu de clasificare a operatorilor îl constituie numărul de operanzi cărora li se aplică. Operatorii pot fi clasificați în operatori **unari** (se aplică la un operand), operatori **binari** (se aplică la doi operanzi) şi **ternari** (se aplică la trei operanzi).

Este de remarcat faptul că există operatori care au dublă semnificație, cum ar fi +, -, *, etc., fiind definiți atât ca operatori unari, cât şi ca operatori binari. Interpretarea se face implicit, în funcție de numărul operanzilor cărora se aplică.

- **Operația unară** se descrie prin sintaxa:

  operator operand;

- **Operația binară** se descrie prin sintaxa:

  operand_1 operator operand_2;

- **Operația ternară** se descrie prin sintaxa:

  operand_1 operator operand_2 operator operand_3;

Operatorii limbajului C++ sunt prezentați în Tabelul nr. 4.1.

Tabelul nr. 4.1. Operatorii limbajului C++

[ ]	**( )**	**.**	**->**	**++**	**--**	**&**	*
+	-	~	**!**	/	**%**	<<	>>
<	>	<=	>=	= =	**!=**	^	|
&&	| |	**? :**	=	* =	/ =	**% =**	+ =
- =	<<=	>>=	**&=**	^ =	| =	**,**	#
##	**sizeof**	**new**	**delete**	**::**	**.***	**->***	

- **Operatorul de atribuire (=)**

Operația cea mai frecvent folosită în programare este operația de atribuire. Prin atribuire, se înscrie (memorează) o valoare într-o variabilă (locație de memorie).

Este operație binară, sintaxa generală de utilizare a operatorului de atribuire fiind:

operand_1 = operand_2;

sau, în mod frecvent:

variabila = expresie;

Pe lângă înscrierea unei valori într-o variabilă, operația returnează valoarea variabilei, astfel că, se pot efectua atribuiri multiple cu sintaxa:

variabila_1 = variabila_2=…= variabila_n = expresie;

În urma atribuirii multiple, toate variabilele din expresie, variabila_1, …, variabila_n, vor avea aceeaşi valoare, cea obținută prin evaluarea expresiei.

```
int a;                  // declarația unei variabile
a=5;                    // atribuirea unei valori constante unei variabile
int b=7;                // declarația unei variabile cu inițializare
a=b;                    // atribuirea unei valori depusă într-o variabilă altei variabile
b=a+10;                 // atribuirea valorii unei expresii unei variabile
a=b=100;                // atribuire multiplă
a=a+b;                  // expresia conține ca operand variabila către care se face
                        // atribuirea; întâi este evaluată expresia a+b cu valoarea pe care
                        // o avea înainte de operație şi apoi se face operația de atribuirea
```

operand_2 (expresie) se mai numeşte rvalue (**right value** - valoare dreapta). Acesta trebuie să fie o expresie care returnează valoare. Expresia poate fi formată din o constantă, o variabilă, apelul unei funcții care returnează valoare, sau o expresie mai complexă.

operand_1 care se găseşte în partea stângă se mai numeşte lvalue (**left value**). Acesta trebuie să desemneze o locație de memorie pentru care există permisiunea de modificare a valorii. Astfel, lvalue poate fi numele unei variabile sau un pointer, dar nu poate fi o constantă, o variabilă declarată const, sau apelul unei funcții.

```
int a = 15;             // declarație de variabilă cu inițializare
const int  ct = 100;    // declarație de variabilă const; inițializarea este obligatorie
ct = 99;                // eroare, se încearcă modificarea valorii unei variabile protejată
                        // prin specificatorul const
```

```
5 = ct;                 // eroare, se încearcă modificarea unei constante literale
a+5 = 20;               // eroare, se încearcă atribuirea unei valori unei expresii
a = getch();            // corect, se atribuie variabilei a valoarea din codul ASCII a
                        // caracterului citit de la tastatură
getch() = a;            // eroare, se încearcă atribuirea unei valori unei funcții
```

Operația de atribuire se poate efectua între tipuri de date diferite. În această situație, se produc conversii implicite ale lvalue către tipul rvalue.

Pot apărea două situații în efectuarea acestor conversii:

- tipul lvalue are reprezentare în domeniu de valori mai larg decât cel al rvalue – aceste conversii nu produc efecte nedorite;
- tipul lvalue are reprezentare în domeniu de valori mai restrâns decât cel al rvalue – aceste conversii se efectuează prin trunchierea valorilor, fapt ce poate duce fie la rotunjirea valorilor, fie la obținerea de valori aleatoare; situațiile sunt descrise în Tabelul nr. 6.2.

Tabelul nr. 4.2. Conversii de tip la atribuire

Tip lvalue	Tip rvalue	Efect conversie
char	int	Se preiau cei mai semnificativi 8 biți
char	unsigned char	Dacă valoarea este >127, se preia valoare negativă
char	long int	Se preiau cei mai semnificativi 8 biți
int	unsigned int	Dacă valoarea este >32767, se preia valoare negativă
int	long int	Se preiau cei mai semnificativi 16 biți
int	float	Se preia partea întreagă a valorii; dacă aceasta depăşeşte domeniul int, se trunchiază
float	double	Se reduce precizia de reprezentare; dacă şi aşa se depăşeşte domeniul float, se trunchiază
double	long double	Se reduce precizia de reprezentare; dacă şi aşa se depăşeşte domeniul long, se trunchiază

Sunt prezentate câteva exemple:

```
char ch = 'a' ;
int a = 10;
char sir[20];
float r =  2.5;
a = 1000;               // atribuire între date de acelaşi tip (int), nu se realizează
                        // conversie
a = ch;                 // conversie cu lărgirea domeniului de valori, a va prelua
                        // valoarea 97 (codul ASCII al caracterului 'a')
ch = a;                 // conversie cu reducere a domeniului de valori, valoarea se
                        // încadrează în domeniul char, deci trunchierea nu va afecta
                        // valoarea preluată
ch = a;                 // conversie cu reducere a domeniului de valori, valoarea nu se
                        // încadrează în domeniul char, deci trunchierea va produce o
                        // valoarea -24
ch = sir;               // eroare, ch este de tip char, sir este adresă de char; cele două tipuri
                        // sunt incompatibile, nu se poate face conversie, deci nici atribuirea
```

- **Operatori aritmetici**

În categoria operatorilor unari se regăsesc atât operatori unari, cât şi binari. Sunt prezentați în Tabelul nr. 4.3.

Tabelul nr. 4.3. Operatori aritmetici

Tip operator	Simbol	Acțiune
Unar	+	Păstrarea semnului, nu produce nici o modificare
	-	Schimbarea semnului
	++	Incrementare
	--	Decrementare
Binar	+	Adunare
	-	Scădere
	*	Înmulțire
	/	Împărțire
	%	Modulo – restul împărțirii întregi

Operatorul + (unar) – aplicat unei expresii va păstra valoarea expresiei.

```
int a;
a = +5;          // expresia +5 întoarce valoarea 5
```

Operatorul - (unar) – aplicat unei expresii va întoarce valoarea expresiei cu semn schimbat.

```
int a, b=2;
a = -b;          // expresia -b întoarce valoarea lui b cu semn schimbat (-2); variabila b
                 // nu este afectată de operație, păstrându-şi valoarea, deci în final a=-2, b=2
```

Operatorii **+ - * /** (binari) efectuează şi returnează rezultatul operațiilor de adunare, scădere, înmulțire, respectiv împărțire.

```
float r = 1.5, s = 3.2, t;
t = r +s;        // operație de adunare
t = r - s;       // operație de scădere
t = r * s;       // operație de înmulțire
t = r / s;       // operație de împărțire
```

Operatorul % (binar) – întoarce restul împărțirii a doi întregi.

```
int a = 15;
int b;
b = a % 2;       // b preia valoarea restului împărțirii lui a la 2, deci preia valoarea 1
b = a % 1.5;     // eroare, unul dintre operanzi nu este întreg.
```

Operatorii ++ şi -- (unari) – sunt operatori de incrementare, respectiv de decrementare.

Operatorul ++ adună valoarea 1 la valoarea operandului. Operatorul – scade valoarea 1 din valoarea operandului. Ambele operații sunt însoțite de înlocuirea valorii operandului cu rezultatul

obținut. Aceasta înseamnă că operandul trebuie să fie o dată de tip lvalue, astfel încât atribuirea să se poată efectua.

```
int a = 5 b = 10;
++a;            // expresia este echivalentă cu: a = a+1; a va primi valoarea 6
--b;            // expresia este echivalentă cu: b = b - 1; b va primi valoarea 9
```

Operatorii de incrementare (++) şi de decrementare (--) pot fi prefixați sau post fixați. În primul caz, modificarea valorii operandului se face înainte de a-i folosi valoarea, în al doilea caz se foloseşte valoarea inițială a operandului şi apoi se efectuează modificarea acesteia.

```
int i = 5, j;
j = ++i                 // operatorul ++ este prefixat, deci întâi se efectuează operația
                        // de incrementare a lui i şi apoi de face atribuirea către j, deci: i=6, j=6;
j = i-- ;               // operatorul -- este postfixat, deci întâi se efectuează operația de
                        // atribuire către j şi apoi de face incrementare a lui i, deci:  i=5, j=6;
```

Aceşti operatori se pot aplica numai datelor de tip lvalue, variabile desemnate prin nume sau pointeri.

```
const int a = 3;
a++;                    // eroare, operatorul de incrementare se aplică unei variabile const
 ++5 ;                  // eroare, operatorul de incrementare se aplică unei constante
(i+j)-- ;               // eroare, operatorul de decrementare se aplică unei expresii
++(ghetchar());         // eroare, operatorul de incrementare se aplică unei funcții
```

- **Operatori relaționali**

Operatorii relaționali sunt operatori binari care se referă la relațiile dintre valorile operanzilor. Aceştia sunt prezentați în Tabelul nr. 4.4.

Operațiile relaționale întorc informație despre faptul că relația între operanzi este adevărată sau falsă.

Limbajele C/C++ nu au tipul de date boolean implementat, ci convenția de reprezentare este următoarea:

- "fals" este reprezentat prin valoarea numerică 0
- "adevărat" este reprezentat prin orice valoare diferită de zero (de regulă valoarea 1).

Tabelul nr. 4.4. Operatori relaționali

Tip operator	Simbol	Acțiune
Binar	>	Mai mare
	>=	Mai mare sau egal
	<	Mai mic
	<=	Mai mic sau egal
	==	Egal
	!=	Diferit

Ca urmare, operațiile relaționale întorc un rezultat care poate avea valoarea 0 sau 1.

```
int a = 5;
float r = 3.5;
a > r;          // expresia întoarce valoarea 1
r >= a;         // expresia întoarce valoarea 0
a == r;         // expresia întoarce valoarea 0
a != r;         // expresia întoarce valoarea 1
```

- **Operatori logici**

Operatorii logici pot fi unari sau binari. Ei realizează operațiile logice între valorile operanzilor. Aceştia sunt prezentați în Tabelul nr. 4.5.

Tabelul nr. 4.5. Operatori logici

Tip operator	Simbol	Acțiune
Unar	!	Negare logică (NOT)
Binar	&&	Şi logic (AND)
	||	Sau logic (OR)

În interpretarea operațiilor logice se are în vedere modul de reprezentare a valorilor logice. Ca urmare, rezultatul acestor operații se poate reprezenta prin tabelele de adevăr prezentate în Fig. 4.1.

p	!p
0	1
1	0

p	q	p && q
0	0	0
1	0	0
0	1	0
1	1	1

p	q	p || q
0	0	0
1	0	1
0	1	1
1	1	1

Fig. 4.1. Tabelele de adevăr pentru operațiile logice

```
int a = 5, b=10;
a && b;          // ambii operanzi sunt diferiți de 0 deci sunt logic adevărați;
                 // rezultatul operației va fi 1
a && (!b);       // (!b) returnează valoarea 0, deci operația && se face între un
                 // operand adevărat şi unul fals, rezultatul va fi 0
a || 3/2;        // operația || se face între doi operanzi diferiți de 0, deci rezultatul va fi 1
```

- **Operatori la nivelul biților**

Operațiile efectuate la nivel de bit apropie limbajele C/C++ de limbajul de asamblare. În mod curent, accesul la memorie se face la nivel de octet, octetul fiind cea mai mică unitate de memorie adresabilă. Prin această categorie de operații se pot face însă operații la nivel de bit. Operațiile la nivel de bit se pot efectua cu operanzi de tip numeric întreg.

Nu se pot efectua astfel de operații cu alte tipuri de date decât întregi, deci nu se pot aplica datelor de tip real (float, double, long double).

Operatorii la nivel de bit pot fi unari sau binari. Operatorii disponibili sunt prezentați în Tabelul nr. 4.6.

Tabelul nr. 4.6. Operatori la nivel de bit

Tip operator	Simbol	Acțiune
Unar	~	Negare la nivel de bit
Binar	&	Și la nivel de bit
	^	Sau exclusiv la nivel de bit
	|	Sau la nivel de bit
	<<	Deplasare la stânga (left shift)
	>>	Deplasare la dreapta (right shift)

```
int a=7, b=2, c;
~ a;                    // expresia returnează valoarea -8
a & b;                  // expresia returnează valoarea 2
a | b;                  // expresia returnează valoarea 7
a ^ b;                  // expresia returnează valoarea 5
a << b;                 // expresia returnează valoarea 28
a >> b;                 // expresia returnează valoarea 1
```

Observații

- Operația de deplasare la stânga (a<<b) este echivalentă cu operația de înmulțire a lui a cu 2b iar Operația de deplasare la dreapta (a>>b) este echivalentă cu operația de împărțire a lui a cu 2b.
- Modul de interpretare a acestor operații reiese din reprezentarea binară a valorilor în memorie.

- **Operatorul *sizeof* – (unar)**

Operatorul sizeof este un operator unar, care are ca rezultat un întreg ce reprezintă numărul de octeți pe care este memorat operandul căruia i se aplică. Operandul este un tip de date sau o dată (constantă sau variabilă) de un anumit tip. Sintaxa de utilizare este:

sizeof (expresie);

```
sizeof(int);            // returnează numărul de octeți pe care este memorat un întreg (2 sau 4)
sizeof(double);         // returnează numărul de octeți pe care este memorat un double (8)
sizeof("ab6de");        // returnează 6, nr. de octeți pe care este memorată constanta
                        // şir "ab6de"
int a
sizeof( a * 1.5);       // returnează numărul de octeți pe care este memorat rezultatul
                        // expresiei a *1.5, care este un double (8)
```

- **Operator & - adresă (unar)**

Operatorul & (unar) a fost menționat în Cursul nr. 03 odată cu introducerea noțiunii de variabilă. Operatorul se aplică numelui unei variabile, rezultatul operației fiind adresa de memorie la care este alocată variabila respectivă. Sintaxa de utilizare este:

&nume_variabila;

```
int a;
&a;                              // returnează adresa la care e alocată variabila a
 printf(“ %p , %u”, &a, &a);  // la execuția instrucțiunii se afişează valoarea adresei
                                 // variabilei a în două formate: hexazecimal (%p) şi întreg
                                 // zecimal fără semn (%u), de exemplu: FFF4, 65524
```

- **Operator condițional ? : (ternar)**

Singurul operator ternar disponibil în C/C++ este operatorul condițional. El este format din caracterele ? şi : şi se aplică la asupra a trei operanzi.

Se foloseşte în expresii sub forma:

e1 ? e2 : e3;

Expresia e1 este evaluată prima. Dacă valoarea acesteia este logic adevărată (non-zero), se evaluează expresia e2 şi aceasta este valoarea returnată de expresia condițională, în caz contrar, se evaluează expresia e3, valoarea acesteia fiind cea returnată.

```
int a, b, c;
scanf(“%d”, &a);
scanf(“%d”, &a);
c = (a>b) ? a : b ;              // c primeşte ca valoare maximul dintre a şi b
```

Expresia anterioară este echivalentă cu secvența:

dacă (a>b) *atunci* c=a *altfel* c=b

sau:

```
if (a>b)                         // if este instrucțiune de decizie care va fi prezentată în cursul
                                 // destinat prezentării instrucțiunilor
        c=a;
else
        c=b;
```

- **Operatorul *(tip)* –conversie explicită de tip (unar)**

Operatorul (tip) este un operator unar care apare în construcții numite cast sau typecast şi converteşte tipul operandului său la tipul specificat între paranteze. Se poate utiliza în orice expresie folosindu-se o construcție de forma:

(nume_tip) expresie

sau

nume_tip (expresie)

Rezultatul întors de operație este valoarea expresiei convertită în mod explicit la tipul de date precizat. Conversia se poate face fie către un tip de dată cu domeniu de valori mai larg decât al expresiei, sau către tip cu reprezentare mai restrânsă decât a expresiei. Expresia se evaluează similar conversiilor produse în mod implicit prin atribuire (vezi Tabelul 4.2.).

```
int a=9;
float r;
(float) a;              // converteşte operandul a (care era de tip întreg) în float
a/(float)2;             // converteşte constanta 2, care este de tip int, la tipul float
r = 9/2;                // se efectuează împărțirea întreagă, 9/2, rezultatul fiind 4 şi apoi se face
                        // atribuirea valorii lui r, astfel, deşi r este de tip float, va prelua valoarea 4
```

```
r = (float)9 / 2;        // 9 este convertit la tipul float, astfel că împărțirea se va face între o
                         // dată float şi o dată int, rezultatul fiind float ; r va primi valoarea 4.5
```

- **Operatorul , – virgulă (binar)**

Operatorul virgulă se foloseşte pentru a obține expresii complexe, alcătuite dintr-o înşiruire de expresii. Sintaxa de utilizare a acestui operator este:

expresie , expresie ;

Valoarea operandului stânga este ignorată, operația returnând ca valoare valoarea expresiei situată în dreapta operatorului.

```
int a = 5, b, c;
c = ( b= 2*a, a+b);      // evaluarea expresiei se face astfel: întâi se atribuie variabilei b
                         // valoarea 2*a, adică 10, apoi se evaluează expresia a+b,
                         // valoarea acesteia, 5+10, fiind valoarea întoarsă de operatorul
                         // virgulă (,), astfel că variabila c primeşte valoarea 15
```

- **Operatorul ( ) – paranteză - operator binar**

Operatorul () se foloseşte pentru a forța ordinea de evaluare a expresiilor. Sunt situații în care ordinea implicită de evaluare a expresiilor nu este convenabilă, situație în care se poate schimba această ordine prin folosirea parantezelor.

```
int a = 2;
int b;
b = a + 2 * a +3;          // ordinea de evaluare este: se efectuează înmulțirea 2*a, apoi
                           // se execută adunările şi în final se face atribuirea, deci b
                           // primeşte valoarea 9
b = (a + 2) *( a +3);      // ordinea de evaluare este: se evaluează adunările, între rezultatele lor
                           // se efectuează înmulțirea şi în final atribuirea, deci b primeşte valoarea 20
```

- **Operatori de atribuire compuşi**

Operatorul de atribuire (=) poate fi combinat cu operatorii aritmetici sau cei la nivel de bit, rezultând o scriere condensată a expresiilor.

Expresii de tipul:

e1 = (e1) operator (e2);

pot fi scrise într-o formă condensată :

e1 operator= e2;

unde (operator=) este operator de asignare în care operator poate fi unul dintre:

+ - * / % << >> & ^ |

Astfel, se pot folosi expresiile extinse sau condensate, ca de exemplu:

i = i+2;	echivalent cu	i+=2;
x= x*(y+1);	echivalent cu	x*=y+1;
a=a>>b;	echivalent cu	a>>=b;

4.2. Ordinea de evaluare a expresiilor

La evaluarea expresiilor, se are în vedere o ordine de efectuare a operațiilor implicită, stabilită pe baza regulilor care stabilesc:

- **prioritatea (precedența)** – indică ordinea de efectuare a operațiilor într-o expresie care conține operatori diferiți;

- **asociativitatea** – indică ordinea de efectuare a operațiilor care se încadrează în aceeaşi grupă de precedență.

Nivelurile de prioritate şi ordinea de evaluare în cazul în care operatori consecutivi intră în alcătuirea aceleiaşi expresii sunt prezentate în Tabelul nr.4.7.

Se observă că operatorii unari au precedență mai mare decât majoritatea operatorilor binari.

Ținând seama de regulile descrise prin Tabelul nr. 4.7., se pot scrie expresii echivalente, cum ar fi:

c = a + b << 2;	echivalent cu	c = (a + b) << 2;
c = a + b + c;	echivalent cu	c = (a + b) + c
a > b && a >c	echivalent cu	(a > b) && (a > c)
~ ! a	echivalent cu	~ (! a)

Tabelul nr. 4.7. Prioritatea şi ordinea de evaluare a operatorilor

Clasa de prioritate	Operatori	Asociativitate
1	( ) [ ] -> :: .	de la stânga la dreapta
2	operatori unari: ! ~ + - ++ -- & * (tip) sizeof new delete	de la dreapta la stânga
3	.* ->*	de la stânga la dreapta
4	* / %	de la stânga la dreapta
5	+ -	de la stânga la dreapta
6	<< >>	de la stânga la dreapta
7	< <= > >=	de la stânga la dreapta
8	= = !=	de la stânga la dreapta
9	&	de la stânga la dreapta
10	^	de la stânga la dreapta
11	|	de la stânga la dreapta
12	&&	de la stânga la dreapta
13	||	de la stânga la dreapta
14	? : (operator condițional)	de la dreapta la stânga
15	= *= /= %= += -= &= ^= |= <<= >>=	de la dreapta la stânga
16	, (operator virgulă)	de la stânga la dreapta

4.3. Conversii de tip în expresii

Conversiile de tip se pot realiza implicit, sau explicit prin folosirea operatorului (tip).

Dacă într-o expresie apar operanzi de tipuri numerice diferite, se fac conversii implicite a operanzilor pentru fiecare operație, astfel încât operația să se efectueze între operanzi de acelaşi tip.

Regula de bază constă în conversia operandului de tip cu domeniu de valori mai mic către tipul cu domeniul de valori mai mare al celuilalt operand. Rezultatul fiecărei operații este de tipul stabilit pentru operanzi.

Excepția o constituie operația de atribuire (=) care respectă regulile descrise în paragraful de descriere a acestui operator. Valoarea membrului drept este convertită la tipul celui stâng. În această situație se pot efectua conversii de la un tip cu reprezentare pe un număr mai mare de octeți la un tip cu reprezentare pe număr mai mic de octeți, ceea ce poate provoca trunchieri (ex.: float -> int), rotunjiri (ex: double -> float) ale valorilor sau pierderea biților de ordin superior în exces (ex: long int -> char).

```
int i = 2;
float r = 3.5;          // valoarea constantă 3.5, care are reprezentare double,
                        // este convertită la tipul float şi apoi este atribuită variabilei r
 (i+2.1)*r              // 2.1 este de tip double, deci valoarea de tip int a variabilei i se converteşte
                        // la tipul double, rezultatul returnat de operația i+2.1 este de tip double, ca
                        // urmare şi valoarea variabilei float r se converteşte la double, rezultatul final
                        // al expresiei fiind şi el de tip double
```

```
int i = 11;
printf("%d", i / 2);                  // afişează 5 – ambii operanzi sunt de tip int, deci şi
                                       // valoarea returnată este de tip  int, astfel că există
                                       // corespondență între format (%d) şi tipul datei afişate
printf("%f", (float) i / 2);           // afişează 5.5 – se impune conversia operandului i la
                                       // tipul float, deci şi operandul 2 se va converti la acelaşi
                                       // tip şi în final tipul valorii returnate este tot float
printf("%f", i/2);                     // afişare eronată – se încearcă afişarea unei valori de tip
                                       // int în format float (%f)
```

Exemple

```
/*************************************************************************
```

Exemplul 4.1. - Se scrie un program în care se exemplifică folosirea operatorilor aritmetici si asimilați cu aceştia: + - * / % ++ -- << >>

```
*************************************************************************/

#include <stdio.h>
#include <conio.h>

void main()
{
 int a, b;

 printf("a=");
 scanf("%d", &a);
 printf("b=");
 scanf("%d", &b);

 printf("\n%d + %d = %d", a, b, a+b);
 printf("\n%d - %d = %d", a, b, a-b);
```

```
  printf("\n%d * %d = %d", a, b, a*b);
  printf("\n%d / %d = %d", a, b, a/b);
  printf("\n%d %% %d = %d", a, b, a%b);
  getch();

  b=++a;
  printf("\nIncrementare (operatorul prefixat): a=%d, b=%d", a, b);
  b=a++;
  printf("\nIncrementare (operatorul postfixat): a=%d, b=%d", a, b);
  getch();

  printf("\n%d << 3 = %d", a, a<<3);
  printf("\n%d >> 3 = %d", a, a>>3);
  getch();
}


/*************************************************************************
```

Exemplul 4.2. - Se scrie un program în care se exemplifică folosirea operatorilor la nivel de bit: ~ & | ^

```
*************************************************************************/
#include <stdio.h>
#include <conio.h>

void main()
{
  int a, b;
  printf("a = ");
  scanf("%d", &a);
  printf("b = ");
  scanf("%d", &b);
  printf("\n ~ %d = %d", a, ~a);
  printf("\n%d & %d = %d", a, b, a&b);
  printf("\n%d | %d = %d", a, b, a|b);
  printf("\n%d ^ %d = %d", a, b, a^b);
  getch();
}



/*************************************************************************
```

Exemplul 4.3. - Se scrie un program în care se exemplifică folosirea operatorilor aritmetici

```
*************************************************************************/

#include <stdio.h>
#include <conio.h>

void main()
{
  int g, m, s;

  printf("grade:");
  scanf("%d", &g);
```

```
  printf("minute =");
  scanf("%d", &m);
  printf("secunde =");
  scanf("%d", &s);

  m += s/60;
  s=s%60;
  g += m/60;
  m = m%60;
  g = g%360;
  printf("\nUnghiul este: %d grd. %d min %d sec.", g, m, s);
  getch();
}
```

```
/*************************************************************************
```

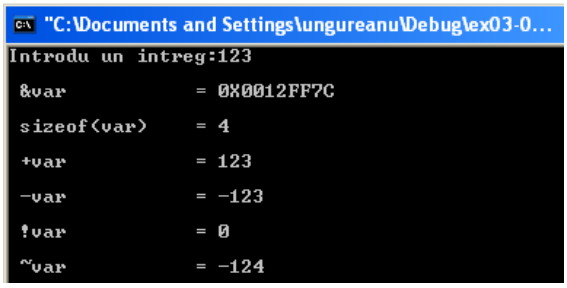
Exemplul 4.4. - Se scrie un program în care se exemplifică folosire a operatorului ternar ? :
- se determina maximul dintre 3 valori.

```
*************************************************************************/
#include <stdio.h>
#include <conio.h>

void main()
{
  int a, b, c,  max;
  printf("a=");
  scanf("%d", &a);
  printf("b=");
  scanf("%d", &b);
  printf("c=");
  scanf("%d", &c);

  //varianta 1
  max = a>b ? a : b;
  max = max>c ? max : c;
  printf("\n Maximul dintre %d, %d, %d este: %d", a, b, c, max);

  //varianta 2
  max = a>b && a>c ? a : b>c ? b : c ;
  printf("\n Maximul dintre %d, %d, %d este: %d", a, b, c, max);
  getch();
}
```

Întrebări. Exerciții. Probleme.

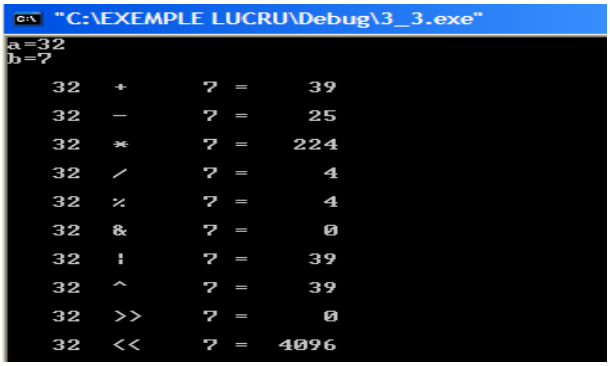
1. Sa se scrie un program in care se declară o variabilă de tip char, una de tip int, una float, una double şi un şir de caractere. Se citesc valori de la tastatură pentru cele cinci variabile. Să se afişeze valoarea, adresa, număr de octeti ocupați pentru fiecare variabilă.

2. Să se scrie un program în care se declară o variabilă de tip int. Se citeşte valoarea variabilei de la tastatură. Să se afişeze rezultatele operațiilor unare care pot fi efectuate cu această variabilă, ca în exemplul prezentat.



3. Să se scrie un program în care se declară o variabilă de tip float. Se citeşte valoarea variabilei de la tastatură. Să se afişeze rezultatele operațiilor unare care pot fi efectuate cu această variabilă (vezi problema nr. 2).

4. Să se scrie un program în care se declară două variabile de tip int. Se citesc valori de la tastatură. Să se afişeze rezultatele tuturor operațiilor binare care pot fi efectuate cu aceste variabile, ca în exemplul prezentat.



5. Să se scrie un program în care se declară două variabile de tip float. Se citesc valori de la tastatură. Să se afişeze rezultatele tuturor operațiilor binare care pot fi efectuate cu aceste variabile, ca în exemplul prezentat.