CURS 4.
Expresii. Variabile

4.1. Expresii

Un *operator* este un simbol care indică compilatorului necesitatea execuţiei unei operaţii matematice sau logice.
O expresie este compusă dintr-un operator si unul sau doi operanzi. Expresiile se evaluează în funcție de regula de evaluare a operatorului implicat in expresie. Rezultatul evaluării unei expresii este o valoare (rezultat al expresiei). In plus, evaluarea expresiei poate să aibă efect şi asupra operanzilor, in sensul modificării valorii acestora.

In funcție de tipul operatorilor, aceştia sunt
- Operatori unari – se aplică asupra unui singur operand
- Operatori binari – se aplică asupra a 2 operanzi.

Expresiile se evaluează fie de la dreapta la stânga, fie de la stânga la dreapta, in funcție de operatorul implicat in expresie.

Expresiile pot fi compuse, in sensul in care rezultatul evaluării unei expresii reprezintă un operand intr-o expresie nouă. In acest sens, operatorii sunt evaluați in funcție de tabela de precedență a operatorilor, descrisă mai jos. Expresiile pot conține () care schimbă prioritatea de evaluare in cadrul expresiilor compuse, adică expresia dintre () se evaluează prioritar.

Precedență	Operator	Descriere	Asociativitate
1	++ -- () [] . –>	Incrementare / decrementare formă postfixată Apel de funcție Referirea unui element dintr-un șir Acces la membrii unei structuri sau uniuni Acces la membrii unei structure sau uniuni prin pointer	De la stânga la dreapta
2	++ -- + – ! ~ (*type*) * & sizeof	Incrementare / decrementare prefixată Plus / minus unar NOT logic și NOT pe biți Conversie de tip Dereferențierea unui pointer Adresa (unei variabile) Mărimea (in octeți a unei variabile / valori)	De la dreapta la stânga
3	* / %	Inmulțire, impărțire, rest	De la stânga la dreapta
4	+ –	Adunare și scădere (operatori binari)	
5	<< >>	Deplasare la stânga/dreapta pe biți	
6	< <= > >=	Operatori relaționali de comparație	
7	== !=	Operatori relaționali pentru egalitate și diferit	
8	&	AND pe biți	
9	^	XOR pe biți (or exclusiv)	
10	|	OR pe biți	
11	&&	AND logic	

12	||	OR logic	
13	?:	Operatorul ternar condițional (singurul operator cu 3 operanzi)	De la dreapta la stânga
14	=	Operatorul de atribuire simplu	
	+= −=	Atribuire cu sumă și diferență	
	*= /= %=	Atribuire cu inmulțire, impărțire și rest	
	<<= >>=	Atribuire cu operații de deplasare pe biți	
	&= ^= |=	Atribuire cu operații pe biți AND , OR sau XOR	
15	,	Operatorul virgulă	De la stânga la dreapta

Descriem mai jos operatorii uzuali utilizați in programele noastre.

Operatori aritmetici:

Operator	Semnificaţie
-	Minus unar (semn)
* / %	Înmulţire, împărţire, restul împărţirii întregi
+ -	Adunare, scădere

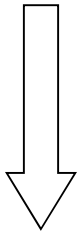
Prioritatea operatorilor scade de sus în jos
Operatorul '%' nu se poate aplica asupra variabilelor de tip float şi double.
Operatorul '/' dacă se aplică la 2 variabile sau constante întregi se va face o împărţire întreagă fără a se lua în considerare restul!

Exemplu:

```
float a=5;
a=a + 1/2;        //variabila float a va conţine în continuare valoarea 5.0
a=a + 1./2;       //în acest caz valoarea variabilei a va fi 5.5
```

Operatori relaţionali şi logici

Operator	Semnificaţie
!	negare logică
> >= < <=	mai mare, mai mare sau egal, mai mic, mai mic sau egal
== !=	egal, diferit
&&	şi logic
||	sau logic



Prioritatea operatorilor scade de sus în jos (deci ! este cel mai prioritar iar || cel mai puţin prioritar).

P	q	!p	p&&q	p||q
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

Observaţii:

- Valoarea de fals în C este reprezentată de valoarea 0. Orice valoare diferită de 0 este interpretată ca valoare de adevărat!
- În urma evaluării expresiilor logice în C rezultatul va fi 0 pentru fals şi 1 pentru adevărat.
- Operatorii din această clasă au prioritate mai mică decât cei aritmetici.

Exemplu: Să se evalueze valoarea de adevăr a expresiei:

10>1+12 → fals (0)

10>8&&!(15<10)||6<=12 ; se evaluează 15<10 ->0 apoi !0 ->1 10>8 are valoarea de adevăr 1 iar 1&&1 conduce spre valoarea de adevăr 1. Evaluarea se opreşte aici deoarece 1||x=1 oricare ar fi x. Deci valoarea de adevăr a întregii expresii este adevărat (1).

Operatori logici pe biţi

Operator	Semnificaţie
&	şi
|	sau
^	xoR (sau exclusiv)
~	complement faţă de 1
>>	decalare la dreapta
<<	decalare la stânga

Operatorii logici pe biţi lucrează la nivel de biţi şi se aplică pe toți octeții variabilelor implicate.

Exemplu:
Să se găsească rezultatul expresiei 7&8.

7 are reprezentarea 0000000000000111
8 are reprezentarea 0000000000001000
Aplicând operatorul şi pe biţi: 0000000000000000

Deci 7&8=0.
Dacă folosim şi logic 7&&8 -> 1 (adevărat şi adevărat → adevărat).

Observaţie: Operatorii relaţionali şi logici generează întotdeauna un rezultat de adevărat sau fals (1 sau 0) în timp ce operatorii pe biţi generează o valoare ce poate fi diferită de 0 sau 1 în funcţie de operaţiile prevăzute.

Operatorii de decalare mută toţi biţii unei variabile spre stânga sau spre dreapta cu un număr de poziţii specificat.

var>>nr_pozitii decalare la dreapta cu numărul de poziţii specificat
var<<nr_pozitii decalare la stânga cu numărul de poziţii specificat

La fiecare decalare spre un capăt, la celălalt capăt se adaugă zerouri.

char x	x după fiecare execuţie	Valoarea lui x
x=7	00000111	7
x=x<<1	00001110	14
x=x<<3	01110000	112
x=x<<2	11000000	192
x=x>>1	01100000	96

x=x>>2	00011000	24

Fiecare decalare la stânga cu o poziţie este echivalentă cu o înmulţire cu 2, pierzându-se bitul cel mai semnificativ. Fiecare decalare la dreapta cu o poziţie este echivalentă cu o împărţire cu 2 şi se pierde bitul cel mai puţin semnificativ.

Exemple:

Se consideră 2 numere întregi n,p unde n€(0,65.535) şi p€(0,15) . Să se seteze pe 1 bitul p din reprezentarea internă a lui n şi să se afişeze noua valoare a lui n.

```
#include<stdio.h>
{
  unsigned int n=5,p=1;
  n|=1<<p;            //operatorul sau se foloseşte pentru setarea pe 1 a unor biţi
                      //având în vedere că x|0=x şi x|1=1 oricare ar fi x
  printf("%u\n"n);
 }
```

Să se seteze pe 0 bitul p din reprezentarea internă a lui n şi să se afişeze noua valoare a lui n;

```
#include<stdio.h>
{
  unsigned int n=7,p=1;
  n&= ~(1<<p); //operatorul şi se foloseşte pentru setarea pe 0 a unor biţi
               // având în vedere că x&0=0 şi x|1=x oricare ar fi x
 printf("%u\n"n);
 }
```

Să se înlocuiască primii p biţi semnificativi din reprezentarea internă a lui n cu complementul lor faţă de 1.

```
#include<stdio.h>
{
  unsigned int n=7,p=3;
  n^= ~0<<(8*sizeof(n)-p);      //vom crea o mască formată din p biţi 1 urmaţi de
                        //biţi 0; în total vom avea 8*sizeof(n) biţi
                        //iar x^0=x şi x^1=!x, oricare ar fi x
 printf("%u\n"n);
 }
```

Operatorul de atribuire

Operatorul de atribuire se utilizează în construcţii de forma:

nume_variabila=expresie;

Acesta are prioritatea cea mai mică. Construcţia de mai sus se numeşte *expresie de atribuire*, fiind un caz particular de expresie. Tipul ei coincide cu tipul lui nume_variabila iar valoarea ei este valoarea atribuită lui nume_variabila. Dacă expresia din dreapta semnului egal are un tip diferit de cel al variabilei nume_variabila atunci întâi valoarea ei se converteşte spre tipul acestei variabile şi pe urmă se realizează atribuirea.

Mesajele de eroare ale compilatorului C precum şi literatura de specialitate foloseşte 2 termeni legaţi de expresia de atribuire: *lvalue* şi *rvalue*. *lvalue* este orice variabilă care poate să apară în partea

stângă a expresiei de atribuire. Practic lvalue reprezintă variabila din expresia de atribuire. *rvalue* se referă la expresia din membrul drept al expresiei de atribuire. Practic rvalue conţine valoarea expresiei.

Este permisă utilizarea unor expresii de atribuire multiplă de forma:

a = b = … = x =expresie;

Pentru operaţia de atribuire, în afara semnului egal, se mai poate folosi şi construcţia op= unde op este un operator care face parte din mulţimea { %, /, *, -, +, &, |, ^, <<, >> }. Folosind această construcţie putem obţine programe C mai scurte, prin comprimarea instrucţiunilor de atribuire. Astfel expresia de atribuire:

v=v op expresie este echivalentă cu v op = expresie.

Construcţie cu operator =	**Exemplu**	**Forma echivalentă**
=	v=10	v=10;
+=	v+=2	v=v+2
-=	v-=a	v=v-a
*=	v*=5	v=v*5
/=	v/=3	v=v/3
%=	v%=2	v=v%2
&=	v&=b	v=v&b
|=	v|=b	v=v|b
^=	v^=a	v=v^a
<<=	v<<=poz	v=v<<poz
>>=	v>>=poz	v=v>>poz

Operatorii de incrementare / decrementare ++ / --

Operatorii de incrementare/decrementare pot să apară în formă prefixată sau postfixată. În forma prefixată putem avea ++v sau --v iar în forma postfixată putem avea v++ sau v--.

Operatorul postfixat (v++) presupune ca expresia să fie evaluată la valoarea (lui v) dinainte de incrementare iar apoi să se realizeze incrementarea operandului.
Operatorul prefixat (de exemplu ++v) presupune ca mai intâi să se realizeze incrementarea lui v iar apoi valoarea incrementată se returnează ca şi valoare a expresiei.

Astfel, instrucţiunea `a=++v` este echivalentă cu secvenţa:

```
v=v+1   (mai intai se face incrementarea)
a=v
```

Instrucţiunea `a=v++` este echivalentă cu secvenţa:

```
a=v  (prima data se face evaluarea expresiei si abia apoi incrementarea)
v=v+1.
```

Exemplu:

```
int a,v=0;
a=++v; //după executarea instrucţiunii v=1 şi a=1
a=v++; //a=1 şi v=2
a=v--; //a=2 şi v=1
a=--v; //a=1 şi v=0;
```

Codul obiect produs de majoritatea compilatoarelor C este mai eficient în cazul utilizării operatorilor de incrementare/decrementare decât cel obţinut prin utilizarea instrucţiunii de atribuire echivalente.

Operatorul de conversie explicită (cast)

Operatorul de conversie explicită este un operator prefixat care se utilizează în construcţii de forma:

```
(tip) operand
```

De multe ori, este necesară forţarea tipului unui operand sau al unei expresii. Prin aceasta, tipul operandului (şi implicit valoarea lui) se converteşte spre tipul indicat între paranteze. Conversiile explicite sunt utile în cazurile în care se doreşte ca rezultatul unei operaţii să fie de alt tip decât cel determinat implicit, pe baza tipurilor operanzilor.

Exemplu: Dacă se doreşte obţinerea rezultatului real, netrunchiat, al împărţirii a 2 numere întregi, cel puţin unul dintre operanzi trebuie convertit explicit la tipul double.

```
int a=10, b=4;
double c;
c=a/b;                  //deşi c este de tip double, el va conţine valoarea 2
c=(float)a/b            //în acest caz c va conţine valoarea 2.5
```

Operatorul dimensiune (sizeof)

Operatorul dimensiune returnează numărul de octeţi ai reprezentării interne a unei date. Acest operator prelucrează tipuri, spre deosebire de ceilalţi operatori care prelucrează valori. Operatorul dimensiune se utilizează în construcţii de forma:

```
sizeof(data)
```

unde data poate să fie variabilă simplă, nume de tablou, tip, element de tablou sau element de structură.

Deoarece tipul operanzilor este determinat încă din faza de compilare, sizeof este un operator cu efect la compilare, adică operandul asupra căruia se aplică sizeof nu este evaluat, chiar dacă este reprezentat de o expresie.

Operatorul condiţional

Operatorul condiţional se utilizează în expresii de forma: `exp1?exp2:exp3;`

Această construcţie are următorul *efect*: Se evaluează valoarea expresiei exp1. Dacă ea are valoarea adevărat, atunci se evaluează exp2, valoarea acesteia fiind şi valoarea întregii expresii. Dacă exp1 are valoarea fals, atunci se evaluează exp3 iar valoarea lui exp3 va fi şi valoarea întregii expresii.

Exemplu:

```
int a=5, b=3,c;
c=a>b?b:a;
```

Se evaluează a>b , adevărat deci c=3.

Construcţia de mai sus este echivalentă cu a scrie:

```
if(a>b)  c=b;
else c=a;
```

Exemple:

Să se scrie programul C care comparând valorile a 2 variabile o afişează pe cea mai mare.

```
#include<stdio.h>
int main()
{ int a=7,b=3;
  printf("maxim=%d\n",a>b?a:b);
}
```

Să se scrie programul C care comparând valorile a 3 variabile o afişează pe cea mai mare.

```
#include<stdio.h>
int main()
{
  int a=7,b=3,c=9;
  printf("maxim=%d\n",a>b?(a>c?a:c):(b>c?b:c));
}
```

Operatorul virgulă

Se utilizează când se doreşte evaluarea mai multor expresii, acestea fiind evaluate de la stânga la dreapta, întreaga expresie luând valoarea ultimei evaluări.

Operatorul virgulă se foloseşte în expresii de forma:

```
expr1, expr2, … exprn
```

Observaţie: Putem utiliza operatorul virgulă pentru a putea permite o serie de instrucţiuni de atribuire, scăpând astfel de necesitatea unei instrucţiuni compuse acolo unde instrucţiunile solicită acest lucru. Întreaga expresie care cuprinde mai multe instrucţiuni de atribuire este considerată o singură instrucţiune.

4.2. Regula conversiilor implicite

O expresie poate conţine operanzi de tipuri diferite. În aceste cazuri, C aplică un sistem de conversii denumit promovarea tipului. Astfel se aplică următoarele reguli:

- Oricare operand de tipul char sau short va fi convertit în int. De asemenea fiecare float este extins spre double prin introducere de zeroruri în partea sa fracţionară. Când un double este convertit spre float, de exemplu printr-o asignare, double-ul este rotunjit înainte de trunchiere pe lungimea unui float.
- Dacă unul dintre operanzi este de tipul double, atunci şi celălalt se converteşte spre tipul double şi rezultatul va fi de tipul double.
- Dacă un operand este long, celălalt este convertit în long şi acesta va fi tipul rezultatului. Dacă un operand este unsigned, celălalt este convertit în unsigned şi acesta va fi tipul rezultatului.

4.3.Variabile

Pentru tratarea corectă a entităţilor identificabile prin nume simbolice (variabile, funcţii, constante simbolice), tipul lor trebuie precizat anterior primei utilizări printr-o declaraţie sau definiţie corespunzătoare. În C toate variabilele trebuie declarate înainte de a fi folosite.

Exemple:

```
char a,b,c;
int d;
double d;
float f;
```

Variabilele se pot defini în 3 locuri:

- în interiorul funcţiilor (locale) ;
- în cadrul definiţiei parametrilor funcţiilor (parametri formali) ;
- înafara oricărei funcţii (globale).

În consecinţă, vorbim de *variabile locale, parametri formali şi variabile globale.*

Variabilele locale (denumite şi automatice) sunt accesibile doar instrucţiunilor care se găsesc în interiorul blocului în care au fost declarate.
În C, obligatoriu toate variabilele locale trebui definite la începutul blocului în care sunt definite, înainte de orice instrucţiune a programului.

Exemplu:

```
void functie()
{ int a;
   a=1;
   int b;        //incorect în C
   b=2;
}
```

Dacă o funcţie urmează să folosească argumente, ea trebuie să declare variabilele pe care le acceptă ca valori ale argumentelor. Aceste variabile sunt denumite *parametri formali* ai funcţiei. Parametri formali pot fi utilizaţi ca variabilele locale obişnuite. La ieşirea din funcţie, acestea sunt distruse la rândul lor.

Exemplu: Să se scrie un program C care calculează 3! şi n!, unde n se citeşte de la tastatură.

```
#include<stdio.h>
int factorial(int n)                          //n este parametru formal şi este declarat de tip întreg
{ int i,fact=1;
   for(i=2;i<=n;i++) fact=fact*i;             //în variabila locală fact calculăm n!
 return(fact);                                //funcţia factorial returnează valoarea lui n!
}
int main()
{ int v;
   printf("3!=%d\n",factorial(3));   //în funcţia printf apelăm funcţia factorial având ca parametru real valoarea 3
   printf("Introd o valoare:");
   scanf("%d",&v);
   printf("%d!=%d\n",v,factorial(v)); //fcţ factorial este apelată având ca param. real val. citită în variabila v
}
```

Variabilele globale se definesc înafara oricărei funcţii. Declaraţia unei variabile globale trebuie plasata înainte de prima utilizare. Este bine ca aceste declaraţii să se facă la inceputul unui fişier cod sursă. Ele sunt utile atunci când mai multe funcţii utilizează aceleaşi variabile.

Exemplu: Să se rescrie programul de mai sus utilizând o variabilă globală.

```
#include<stdio.h>
int fact=1;                          //se defineste variabila globală fact de tip întreg şi se da val 1
void factorial(int n)
{ int i;
  fact=1;
  for(i=2;i<=n;i++) fact=fact*i; //variabila globală fact va conţine valoarea lui n!
}
int main()
{ int v;
  factorial(3);              //apelăm funcţia factorial cu un parametru
                             //real având valoarea 3
  printf("3!=%d\n",fact);          //afişăm valoarea lui 3!; se observă că valoarea
```

```
                                  //variabilei globale fact este cunoscută atât în
                                  //funcţia main cât şi în funcţia factorial
 printf("Introd o valoare:");
  scanf("%d",&v);
  factorial(v);                   //apelăm din nou funcţia factorial pentru valoarea v
 printf("%d!=%d\n",v,fact); //afişăm valoarea lui v!; din nou folosim
                                  //variabila globală fact
}
```

Astfel putem defini *variabile globale* care pot fi accesate de toate funcţiile care sunt definite după instrucţiunea de definire a acesteia. Spre deosebire de variabilele locale, care se distrug în momentul părăsirii funcţiei, variabilele globale rămân disponibile pentru utilizare pe toată durata de execuţie a programului. Definiţiile pot sa apară oriunde în interiorul unui program, dar funcţiile definite înaintea lor nu le vor recunoaşte.

Practica tradiţională în C utilizează literele mici pentru nume de variabile şi literele mari pentru constante simbolice.

Variabile de tip tablou

Pentru tablouri se folosesc construcţii de forma:

```
tip lista_de_elemente;
```

unde element are forma nume[lim1][lim2]… [lim n]. lim1, lim2 … lim n reprezintă numere întregi sau expresii constante (expresii a căror valoare poate fi evaluată în faza de compilare).

Observaţie importantă: Indicii în C pornesc de la 0!

Exemple:

```
int a[10];        //se declară un tablou de 10 numere întregi
float b[5][4];    //se declară un tablou de numere reale având 5 linii şi 4 coloane
char c[4];        //se declară un şir de caractere ce poate conţine maxim 3 caractere
```

Definiţii / declaratii de variabile

Pentru o variabilă, putem avea o singură definiţie si oricate declaraţii. Pentru variabilele locale, definiţiile de variable sunt in acelaşi timp şi declaraţii şi implică alocarea variabilelor respective in zona de stivă a memoriei.

Pentru variabilele globale, definiţia implică alocare de memorie pentru variabilă în zona globală de date a programului. Pentru ca o variabilă globală să poată fi utilizată (de exemplu intr-un alt fisier decât in cel unde a fost definită) se utilizează cuvântul cheie extern inainte de definiţie.

Definiţia unei variabile poate fi completată prin specificarea unei valori iniţiale şi atunci se spune că am initiaţizat variabila respectivă.

Momentul efectuării iniţializării precum şi eventualele iniţializări implicite sunt condiţionate de clasa de memorare în care este inclusă variabila respectivă.

- Variabilele alocate in zona globală de date a programului se iniţializează o singură dată, înaintea începerii execuţiei programului, cu valoarea declarată, sau, în absenţa acesteia cu valoarea implicită zero.
- Variabilele din clasa *automatic* (locale funcţiilor) sunt iniţializate numai dacă s-a cerut explicit, iniţializarea fiind reluată la fiecare apel al funcţiei în care au fost definite.

Variabilele automatice sunt locale fiecărei apelări a unui bloc sau unei funcţii şi sunt declasate (îşi pierd declaraţia) la ieşirea din blocul respectiv.

Dacă nu sunt iniţializate, aceste variabile conţin valori reziduale. Nici o funcţie nu are acces la variabilele din altă funcţie. În funcţii diferite, pot exista variabile locale fără ca acestea să aibă vreo legătură între ele.

În absenţa iniţializării explicite, variabilele globale sunt iniţializate implicit cu valoarea 0 în timp ce variabilele locale au valori iniţiale nedefinite (reziduale).

O caracteristică a limbajului C este că o declaraţie se poate referi la tipuri diferite, derivate din acelaşi tip de bază. Astfel, declaraţia:

```
char c, *p, sir[10];
```

se referă la trei variabile de tipuri diferite: caracterul c, pointerul la caracter p şi şirul de caractere sir pentru care se rezervă 10 octeţi.

4.4. Comentarii

Un comentariu începe cu succesiunea de caractere /* şi se termină */ în cadrul comentariului neavând voie să apară aceste construcţii (deci comentariile nu pot fi imbricate). Comentariile pot fi plasate oriunde în program, atât timp cât ele nu apar în mijlocul unui cuvânt cheie sau identificator.
ÎSe poate introduce un comentariu linie folosind succesiunea de caractere ‘//’. Datorită simplităţii celei de a doua forme, în exemplele prezentate am folosit comentarii linie, cu menţiunea că acestea sunt specifice limbajului C++.

Exemple:

```
/* Acesta este
un comentariu in C*/
int tablou[10][10];     //acesta e un comentariu linie C++; se declară un tablou
                       //bidimensional
```