Programarea Calculatoarelor

Cursul 2: Reprezentarea internă a datelor. Expresii și operatori

Ion Giosan
Universitatea Tehnică din Cluj-Napoca
Departamentul Calculatoare

Baze de numerație (1)

- Transformarea unui număr natural din baza 10 în baza 2
  - Exemplu
    - 157:2=78+ **1**
    - 78:2 =39+ **0**
    - 39:2 =19+ **1**
    - 19:2 =9 + **1**
    - 9:2 =4 + **1**
    - 4:2 =2 + **0**
    - 2:2 =1 + **0**
    - 1:2 =0 + **1**

$157_{(10)}=10011101_{(2)}$

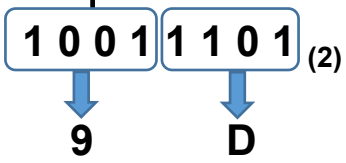
- Transformarea unui număr natural din baza 2 în baza 10
  - Exemplu

7	6	5	4	3	2	1	0
1	**0**	**0**	**1**	**1**	**1**	**0**	**1**

$10011101_{(2)}=1*2^0+0*2^1+1*2^2+1*2^3+1*2^4+0*2^5+0*2^6+1*2^7=157_{(10)}$

Baze de numerație (2)

- Transformarea unui număr natural din baza 2 în baza 16 : grup de câte 4 cifre binare => o cifră hexazecimală
  - Exemplu



$10011101_{(2)}=9D_{(16)}$

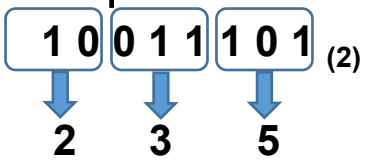
  - Cifrele hexazecimale: 0…9 și A…F (pentru valorile de la 10 la 15)
- Transformarea unui număr natural din baza 16 în baza 10
  - Exemplu

1	0
9	**D**

$9D_{(16)}=13*16^0+9*16^1=157_{(10)}$

Baze de numerație (3)

- Transformarea unui număr natural din baza 2 în baza 8 (octal) : grup de 3 cifre binare => o cifră în octal
  - Exemplu



$10011101_{(2)}=235_{(8)}$

- Transformarea unui număr natural din baza 8 în baza 10
  - Exemplu

2	1	0
2	**3**	**5**

$$235_{(8)}=5*8^0+3*8^1+2*8^2=157_{(10)}$$

Reprezentarea internă a datelor (1)

- Reprezentarea cu mărime și semn
  - Pentru un număr reprezentat pe **n** biți numerotați de la **0** la **n-1**:

a_{n-1}	a_{n-2} ... ... a_0
Semn	**Valoare**

  - Numărul reprezentat este:

$$X = (-1)^{a_{n-1}} \sum_{k=0}^{n-2} a_k \cdot 2^k$$

  - Exemplu:
    - 157 este reprezentat pe 32 biți ca
      - 0000 0000 0000 0000 0000 0000 1001 1101 în baza 2
      - 00 00 00 9D în baza 16
    - -157 este reprezentat 32 biți ca
      - 1000 0000 0000 0000 0000 0000 1001 1101 în baza 2
      - 80 00 00 9D în baza 16
  - Ar exista două reprezentări pentru valoarea zero!
    - 0000…0 și 1000…0

Reprezentarea internă a datelor (2)

- Reprezentarea în memorie a numerelor întregi cu semn
  - Tipurile de date: **char, short, int, long int, long long int**
  - Codul complement față de 2 (**C2**)
    - Numerele pozitive se reprezintă identic ca și în reprezentarea mărime și semn
    - Numerele negative:
      - Se calculează complementul față de 1 (**C1**) al valorii absolute prin inversarea tuturor biților
      - La această valoare a lui **C1** se adaugă **1**
  - Exemple
    - 157 este reprezentat în C2 pe 32 biți ca
      - 0000 0000 0000 0000 0000 0000 1001 1101
      - 00 00 00 9D $_{(16)}$
    - -157 este reprezetat în C2 pe 32 biți ca
      - 1111 1111 1111 1111 1111 1111 0110 0011
      - FF FF FF 63 $_{(16)}$
      - Calculul reprezentării în C2 a lui -157:

        0000 0000 0000 0000 0000 0000 1001 1101 (valoarea absolută 157)

        1111 1111 1111 1111 1111 1111 0110 0010 (C1)

        1111 1111 1111 1111 1111 1111 0110 0011 (C2)

Reprezentarea internă a datelor (3)

- Reprezentarea în memorie a numerelor întregi cu semn
  - Numărul reprezentat pe cei **n** biți este:

$$X = -a_{n-1} \cdot 2^{n-1} + \sum_{k=0}^{n-2} a_k \cdot 2^k$$

  - Domeniul de valori [-2^{n-1}, $2^{n-1}-1$]
  - O singură reprezentare pentru zero: 0000…0
- Reprezentarea în memorie a numerelor întregi fără semn
  - Tipurile de date: **unsigned char, unsigned int**, etc.

a_{n-1} ... … … … … a_0
Valoare

  - Numărul reprezentat este:

$$X = \sum_{k=0}^{n-1} a_k \cdot 2^k$$

  - Domeniul de valori [0, 2^n-1]

Reprezentarea internă a datelor (4)

- Reprezentarea în memorie a numerelor reale (IEEE-754)
  - Tipurile de date: **float, double, long double**
  - Pentru un număr reprezentat pe **n** biți numerotați de la **0** la **n-1**:

a_{n-1}	a_{n-2} ... ... a_k	a_{k-1} ... ... a_0
Semn	**Exponent**	**Mantisă**

  - Numărul reprezentat este:

$$X = (-1)^{semn} \times 2^{exponent-deplasament} \times 1.mantisă$$

  - Pentru tipul **float**:
    - 1 bit *semn*, 8 biți *exponent*, 23 biți *mantisă*
    - *deplasament* = 127; în baza 2: 01111111
  - Pentru tipul **double**:
    - 1 bit *semn*, 11 biți *exponent*, 52 biți *mantisă*
    - *deplasament* = 1023; în baza 2: 01111111111

Reprezentarea internă a datelor (5)

- Reprezentarea numerelor reale (formatul IEEE-754)
  - Exemple (http://www.binaryconvert.com/)
  - Numărul 132.57 stocat pe tipul **float**

a_{31}	a_{30} ... … … … a_{23}	a_{22} ... … … … … … … a_0
0	**10000110**	**00001001001000111101100**

    - Conținutul în hexazecimal: 0x430491EC
    - $132.57=(-1)^0 \times 2^{10000110-01111111} \times 1.00001001001000111101100$
  - Numărul -132.57 stocat pe tipul **double**

a_{63}	a_{62} ... … a_{52}	a_{51} ... … … … … … … … … … … … … … … a_0
1	**10000000110**	**0000100100100011110101110000101000111101011100001010**

    - Conținutul în hexazecimal: 0xC060923D70A3D70A
    - $-132.57=(-1)^1 \times 2^{10000000110-01111111111} \times$ $1.0000100100100011110101110000101000111101011100001010$

Reprezentarea internă a datelor (6)

- Consecințe
  - Numerele întregi
    - Sunt reprezentate exact
    - O valoare întreagă care depășește un capăt al domeniul de reprezentare al numerelor întregi se găsește la extrema cealaltă a domeniului la o distanță egală cu valoarea depășită
  - Numerele reale
    - Nu pot fi întotdeauna reprezentate exact
    - O valoare reală care depășeste capătul inferior al domeniului de reprezentare este tratată ca fiind **minus infinit**
    - O valoare reală care depășeste capătul superior al domeniului de reprezentare este tratată ca fiind **plus infinit**

Reprezentarea internă a datelor (7)

- Consecințe – exemplu

```
#include <stdio.h>
#include <float.h>
int main() {
      int x = 0x7FFFFFFF, y = x+10;
      printf("%d %d\n", x, y); //2147483647 -2147483639
      double a = 0.1, b = 0.2, c = 0.3;
      printf("%.20f %.20f %.20f\n", a, b, c);
    //0.10000000000000001000 0.20000000000000001000 0.29999999999999999000
      printf("%d %d\n", a+b==c, c==c); // 0 1
      float e = 30000000, f = 1, g = e+f;
      printf("%.15f %.15f %.15f\n", e, f, g);
    //30000000.000000000000000 1.000000000000000 30000000.000000000000000
      double t=DBL_MAX, u=t*1.000001;
      printf("%f", u); //inf
      return 0;
}
```

Expresii (1)

- **Expresia** reprezintă mai mulți **operanzi** legați prin **operatori**
- Un **operand** poate fi
  - O constantă numerică sau caracter
  - O constantă simbolică
  - Un identificator de variabilă simplă
  - Un identificator de variabilă tablou
  - Un identificator de structură
  - Un identificator de tip
  - Un identificator de funcție
  - O variabilă indexată
  - O componentă a unei structuri
  - Un apel de funcție
  - O expresie scrisă între paranteze rotunde

Expresii (2)

- Un **operand** are
  - **Tip**
  - **Valoare**
- Un **operator** poate fi
  - **Unar**
    - Aplicat unui singur operand
  - **Binar**
    - Aplicat operandului care îl precedă și celui care îl succedă

Operatori (1)

- Clase de operatori
  - Aritmetici
    - Operatorii unari **+ –**
    - Operatorii binari **+ - * / %**
  - Relaționali **< <= > >=**
  - De egalitate **== !=**
  - Logici **! && ||**
  - Pe biți
    - Logici **~ & | ^**
    - De deplasare **<< >>**
  - De asignare
    - Operatorul de asignare simplă **=**
    - Operatorul de asignare compusă **op=**
      - **op** este un operator aritmetic sau logic pe biți
      - Asignarea compusă **v op = operand** este echivalentă cu **v = v op operand**
  - De incrementare **++** și decrementare **--**

Operatori (2)

- Clase de operatori (continuare)
  - De conversie de tip **(tip) operand**
  - De dimensiune
    - **sizeof(tip)**
    - **sizeof(operand)**
  - Operatorul adresă **&operand**
  - Operatorul de dereferențiere ***operand**
  - Operatorii paranteză **( ) [ ]**
  - Operatorul condițional **?**
    - **expresie ? expresie1 : expresie2**
  - Operatorul virgulă **,**
  - Operatorii de acces la elementele unei structuri sau uniuni
    - **.** cand operandul este o structură sau uniune
    - **->** cand operandul este pointer la o structură sau uniune (adresa de memorie unde este stocată acea structură sau uniune)

Operatori aritmetici (1)

- Unari **+** și **–**

```
int a, b=-5; // b este -5; a este doar declarat
int a=+b;    // a este acum tot -5
```

- Binari: adunare **+** scădere **–** înmulțire * împărțire **/** modulo **%**

```
int a=13, b=5;
int c=a+b; // c este 18
c=a-b; // c este 8
c=a/b; // c este 2 (câtul împărțirii)
c=a%b; // c este 3 (restul împărțirii)
       // 13/5 = 2 rest 3
float d=a/b; // d este 2
d=a/5.0f;    // d este 2.6
```

Operatori aritmetici (2)

- Împărțirea la zero
  - Eroare în cazul tipului întreg

```
int a=13;
int b=a/0; //eroare la execuția programului
```

  - Plus (**inf**) sau minus infinit (**-inf**) în cazul tipurilor reale

```
float x=13;
float y=x/0;  //y este inf
float z=-x/0; //z este -inf
```

- Unele operații între plus și minus infinit pot da rezultat "not-a-number" (**nan**)

```
/* y si z sunt cei din exemplul anterior */
float t=y+z; //t este nan
float w=y*z; //w este -inf
float s=y/z; //s este nan
```

Operatori relaționali și de egalitate

- Operatorii mai mic **<** mai mic sau egal **<=** mai mare **>** mai mare sau egal **>=**
- Operatorii egal **==** diferit **!=**
- Rezultatul expresiei este o valoare logică
  - 0 dacă este fals
  - 1 dacă este adevărat

```
int a=13;
double b=3.5;
int c=a<b; //c este 0
c=(a>=b);  //c este 1
c=(a==b);  //c este 0
c=(a!=b);  //c este 1
c=(a>b>2.7); /* c este 0, a>b se evalueaza mai intai
                la 1 iar apoi se evalueaza expresia
                1>2.7 la fals, adica 0 */
```

Operatori logici (1)

- În C toate **valorile diferite de zero** sunt **logic adevărate**, doar **valoarea zero** înseamnă **logic fals**
- Operatorul negare logică **!**
  - Operator unar care neagă o expresie logică
  - **!expresie** este 0 dacă **expresie** are o valoare adevărată (nenulă) și 1 dacă **expresie** este falsă (adică are valoarea zero)
- Operatorul ȘI logic **&&**
  - Operator binar care realizează operația ȘI logic
  - **expresie1 && expresie2** este 1 numai în cazul în care ambele expresii sunt adevărate (sunt nenule); 0 altfel
- Operatorul SAU logic **||**
  - Operator binar care realizează operația SAU logic
  - **expresie1 || expresie2** este 1 în cazul în care cel puțin una din cele două expresii sunt adevărate (are valoare nenulă); 0 altfel

Operatori logici (2)

- Într-o expresie logică mai complexă, dacă valoarea finală se poate deduce la un moment dat atunci restul nu se mai evaluează (***scurt-circuit***)
- Exemple

```
int a=150;
float b=-2.5f;
int c=!b;   //c este 0
int d=a&&c; //d este 0
int e=a||c; //e este 1
int f=a&&d&&(d=b); /* f este 0, d rămâne 0,
                      expresia d=b nu s-a
                      mai evaluat */
```

Operatori logici pe biți

- Complement față de 1 **~**
  - Neagă toți biții din reprezentarea internă a operandului căruia i-a fost aplicat
- ȘI pe biți **&**
  - Realizează operația de ȘI logic între biții din reprezentarea internă a celor doi operanzi între care se aplică
- SAU pe biți **|**
  - Realizează operația de SAU logic între biții din reprezentarea internă a celor doi operanzi între care se aplică
- SAU-EXCLUSIV pe biți **^**
  - Realizează operația de SAU-EXCLUSIV logic între biții din reprezentarea internă a celor doi operanzi între care se aplică
  - Rezultatul operației SAU-EXCLUSIV între doi biți este 1 doar dacă unul dintre biți este 1 iar celălalt este 0; altfel rezultatul este 0
- Exemple

```
int a=13; // reprezentarea pe biti este 000…01101
int b=10; // reprezentarea pe biti este 000…01010
int c=~a; /* c este in baza 2 : 111…10010
             c este in baza 16: FFFFFFF2
             c este in baza 10: -14        */
int d=a&b; // d este in baza 2: 000…01000; in baza 10: 8; in baza 16: 8
int e=a|b; // e este in baza 2: 000…01111; in baza 10: 15; in baza 16: F
int f=a^b; // f este in baza 2: 000…00111; in baza 10: 7; in baza 16: 7
```

Operatori de asignare (1)

- Asignare simplă **=**
  - **expresie1=expresie2**
  - **expresie1** trebuie să permită stocarea valorii **expresie2** în memorie
  - Dacă tipul lui **expresie1** nu este același cu cel al lui **expresie2**, atunci valoarea lui **expresie2** se convertește în momentul asignării la **expresie1** (conversie implicită)
  - Operatorul poate fi folosit în înlănțuiri și este asociativ dreapta

- Exemple

```
int 13=a; // nu este permis !!!
int a=13; // corect, a este 13
int b=10; // b este 10
int c=b; // c este 10
int d=c=a+2; //c este 15, d este tot 15
```

Operatori de asignare (2)

- Asignare compusă **op=**
  - **op** este un operator aritmetic sau logic pe biți
  - Asignarea compusă **v op = operand** este echivalentă cu **v = v op operand**

- Exemple

```
int a=13; // a este 13
a*=10; // echivalent cu a=a*10; a este 130
a<<=1; // echivalent cu a=a<<1; a este 260
a|=5; // echivalent cu a=a|5; a este 261
```

Operatori de deplasare pe biți

- Se aplică între operanzi întregi în care cel de-al doilea operand trebuie să fie număr natural mai mic decât numărul de biți pe care este reprezentat primul operand
- Deplasare spre stânga **<<**
  - Deplasează spre stânga biții din reprezentarea internă a primului operand cu un număr de poziții egal cu cel de-al doilea operand
  - Se inserează în partea dreaptă un număr de biți de 0 egal cu valoarea deplasamentului
  - Este echivalent cu o înmulțire a primului operand cu 2 la puterea deplasamentului
- Deplasare spre dreapta **>>**
  - Deplasează spre dreapta biții din reprezentarea internă a primului operand cu un număr de poziții egal cu cel de-al doilea operand
  - Se inserează în partea stângă un număr de biți de valoarea bitului de semn (primul bit) egal cu valoarea deplasamentului
  - Este echivalent cu o împărțire a primului operand cu 2 la puterea deplasamentului
- Exemple

```
int a=13;  // reprezentarea pe biti este 000…01101
int b=-14; // reprezentarea pe biti este 111…10010
int c=a<<3; // c este pe biti:000…01101000; in baza 10: 104
int d=b>>2; // d este pe biti:111…11111100; in baza 10: -4
```

Operatori de incrementare și decrementare (1)

- Operatorul de incrementare **++**
  - Realizează incrementarea unei variabile cu 1 și returnează valoarea dinainte sau de după incrementare astfel:
  - Forma prefixă **++i**
    - Pre-incrementează variabila i cu 1
    - Returnează valoarea **i+1**, adică valoarea incrementată
  - Forma postfixă **i++**
    - Returnează valoarea **i**, adică valoarea dinainte de incrementare
    - Post-incrementează variabila i cu 1
- Exemple

```
int a=13;
int b=a++; // b este 13, iar a este 14
int c=++a; // c este 15, iar a este tot 15
a++; // a este 16
++a; // a este 17
```

Operatori de incrementare și decrementare (2)

- Operatorul de decrementare **--**
  - Realizează decrementarea unei variabile cu 1 și returnează valoarea dinainte sau de după decrementare astfel:
  - Forma prefixă **--i**
    - Pre-decrementează variabila i cu 1
    - Returnează valoarea **i-1**, adică valoarea decrementată
  - Forma postfixă **i--**
    - Returnează valoarea **i**, adică valoarea dinainte de decrementare
    - Post-decrementează variabila i cu 1
- Exemple

```
int a=15, b=10, c;
a--; //a este 14
--a; //a este 13;
c = b--; //c este 10, b este 9
c = ++a - b--; //a este 14, c este 5, b este 8
c = a-- + ++b; //b este 9, c este 23, a este 13
```

Operatorul de conversie de tip (*type cast*)

- Realizează conversia tipului unui operand doar în locul în care acesta este implicat într-o expresie
  - Se scrie noul tip la care se realizează conversia, între paranteze rotunde, înaintea operandului asupra căruia se aplică:

    (tip) x
- Observație:
  - Tipul unei variabile nu poate fi modificat permanent, acesta rămânând cel care i-a fost atribuit variabilei în momentul declarării
- Exemple

```
int a=27, b=10;
float c = a/b; //c este 2
c = a / (float) b; //c este 2.7
b = c; // conversie implicită la tipul int; b este 2
b = (int) c; // conversie explicită de tip; b este 2
float d = a + c; // d este 29.7
double e = ((double)a) / 2; // e este 13.5
double e = a / 2.0; // e este 13.5
```

Operatorul de dimensiune

- Operatorul **sizeof**
  - **sizeof(tip)**
    - Returnează numărul de octeți ocupați în memorie de o variabilă de tipul **tip**
  - **sizeof(operand)**
    - Returnează numărul de octeți ocupați în memorie de **operand**
- Exemple

```
int a=27;
float b[10];
double c[2][15];
a = sizeof(a); // a este 4
a = sizeof(char); // a este 1
a = sizeof(float); // a este 4
a = sizeof(double); // a este 8
a = sizeof(b); // a este 40
a = sizeof(c); // a este 240
```

Operatorul adresă

- Operatorul **&**
  - Scris în fața unei variabile determină adresa de memorie unde este stocată variabila respectivă
    - Este un pointer la acea variabilă (mai multe detalii în cursul dedicat pointerilor)
    - Orice adresă validă este o valoare diferită de 0 (adresa 0 sau NULL reprezintă o adresă invalidă; un pointer de valoare 0 sau NULL nu referă nimic)
- Exemplu
  - Utilizare pentru citirea variabilelor simple cu format

```
int a;
float b;
double c;
scanf("%d %f %lf", &a, &b, &c);
printf("Valorile citite sunt: %d %f %f", a, b, c);
```

Operatorii paranteză

- Paranteze rotunde **( )**
  - Realizează gruparea unor expresii cu scopul de a fi evaluate înaintea altor expresii
  - Exemple

```
int a=27, b=3;
float c=10;
float d = a - b / c; // d este 26.7
d = (a - b) / c; // d este 2.4
```

- Paranteze drepte **[ ]**
  - Permit declararea de tablouri și accesul la elementele acestora
  - Exemple

```
int a[5]={40,-20};
char b[5][2]={{'A','b'},{'0','a'}};
int c = a[1]; // c este -20
char d = b[0][1]; // d este 'b' sau 98
d = b[1][0]; // d este '0' sau 48
int e = b[1][1]; // e este 'a' sau 97
```

Operatorul condițional

- Operator ternar

  expresie ? expresie1 : expresie2

  - Asemănător instrucțiunii **if**
  - Returnează valoarea **expresie1** dacă **expresie** este adevărată
  - Returnează valoarea **expresie2** dacă **expresie** este falsă
- Exemple

```
int a=7;
(a%2) ? printf("numar impar") : printf("numar par");
     /* se afiseaza textul "numar impar"
        valoarea returnata este 11, adica numarul de
        caractere afisat, dar aceasta nu este retinuta
        in nicio variabila
     */
int b = (a>10) ? (a/7) : (a*7); // b este 49, a este 7
```

Operatorul virgulă

- Permite evaluarea secvențială a unei expresii compuse din mai multe expresii separate prin virgule
  - Evaluarea se face de la stânga la dreapta
  - Valoarea ultimei expresii din înlănțuire este valoarea expresiei compuse
- Exemple

```
int a=7;
char b='C';
float c=3.9;
a = (c/2, b++, b-2);
printf("a=%d\nb=%c\nc=%f", a, b, c);
     /* se afiseaza:
          a=66
          b=D
          c=3.9     */
```

Evaluarea expresiilor

- La evaluarea unei expresii se ține cont de
  - Prioritatea operatorilor
  - Asociativitatea operatorilor în caz de prioritate egală
  - Regulile de conversie implicită
- Rezultatul evaluării unei expresii în care operanzii sunt de același tip va avea același tip de date cu cel al operanzilor
  - Excepție: operanzi de tip întreg cu rang mai mic ca **int** într-o expresie sunt promovați la **int**
- Observație
  - În cazul în care nu cunoaștem bine prioritatea operatorilor, expresia care dorim să fie evaluată mai întâi se va specifica între paranteze rotunde **( )**

Prioritatea și asociativitatea operatorilor

Prioritate	Operatori	Simboluri	Asociativitate
1	Incrementare decrementare postfixată Apel de funcție Acces la elementele unui tablou Acces la elementele unei structuri sau uniuni direct sau prin pointer	++ -- () [] . ->	stânga-dreapta
2	Incrementare decrementare prefixată Aritmetici unari Negare logică și pe biți Conversie de tip De dereferențiere Adresă Dimensiune	++ -- + - ! ~ (tip) * & sizeof	dreapta-stânga
3	Înmulțire, împărțire, rest	* / %	stânga-dreapta
4	Adunare, scădere	+ -	stânga-dreapta
5	Deplasare stânga și dreapta pe biți	<< >>	stânga-dreapta

Prioritatea și asociativitatea operatorilor

Prioritate	Operatori	Simboluri	Asociativitate
6	Relaționali	< <= > >=	stânga-dreapta
7	De egalitate	== !=	stânga-dreapta
8	ȘI pe biți	&	stânga-dreapta
9	SAU-EXCLUSIV pe biți	^	stânga-dreapta
10	SAU pe biți	|	stânga-dreapta
11	ȘI logic	&&	stânga-dreapta
12	SAU logic	||	stânga-dreapta
13	Condițional	? :	dreapta-stânga
14	Asignare simplă Asignare compusă	= += -= *= /= %= <<= >>= &= ^= |=	dreapta-stânga
15	Virgulă	,	stânga-dreapta

Regulile de conversie implicită (1)

- La atribuire
  - Valoarea expresiei din dreapta operatorului de atribuire se convertește automat la tipul identificatorului din stânga
    - Conversia de la un tip real la un tip întreg se face prin trunchiere (tăierea părții zecimale de după virgulă) !
    - Se poate întâmpla ca tipul de date la care se face conversia să nu poată reprezenta valoarea convertită !

```
double t=878.598;
int e = t;
char f = t;
printf("%d, %d", e, f); // 878, 110
```

- Conversia implicită a operanzilor implicați într-o operație se face la tipul de date a celui cu rang mai înalt din ierarhie (care poate reprezenta ambele valori)
  - Ordinea descrescătoare a rangului tipului de date:
    - long double, double, float, long long, long, int, char

```
int i=27, j=10;
float u = i / j; // u este 2 de tip float
double v = i / (float) j; // v este 2.7 de tip double
printf("%g, %g", u, v); // 2, 2.7
```

Regulile de conversie implicită (2)

- Pentru același tip de date întreg, varianta fără semn are rang mai mare decât varianta cu semn

```
unsigned int a = 3000000000;
int b = 1;
unsigned int c = a + b;
printf("%u\n", c); // 3000000001
```

- În cazul în care toți operanzii sunt întregi se efectuează promovarea întregilor
  - Tipurile de date cu rang mai mic decât **int** cum sunt tipurile de date **char** și **short** sunt promovate la **int**
    - În cazul variantei fără semn (**unsigned**) acestea sunt promovate la **unsigned int**

```
char x=120;
char y=110;
int z = x + y;
printf("%d\n", z); // 230
```