Fundamentele programării

Curs 3

Instrucțiuni. Operatori.

Elena BAUTU & Dorin-Mircea POPOVICI
{ebautu,dmpopovici}@univ-ovidius.ro

Cuprins

- Instructiuni de ciclare
  - for,
  - while,
  - do-while
- Alte instructiuni
  - break; continue; goto
- Operanzi & Operatori
  - Tipuri
  - Prioritate

Instructiuni de ciclare

Instrucţiunea de ciclare do-while (cont)

execuţă o instrucţiune cât timp o expresie are valoare nenulă (adevărată)

- sintaxa:

```
do{
instructiune/i;
}while(conditie);
```

- efect:
  - se execută instructiune;
  - dacă valoarea expresiei conditie este 0 (fals), atunci se opreşte execuţia lui do-while,
  - altfel se repetă încă o dată corpul lui while.

Instrucţiunea de ciclare do-while (cont)

- Se dorește citirea unui număr între 1 și 10. Dacă utilizatorul introduce un număr care nu satisface această condiție, atunci i se cere din nou.

Instrucţiunea de ciclare do-while (cont)

- Se dorește citirea unui număr între 1 și 10. Dacă utilizatorul introduce un număr care nu satisface această condiție, atunci i se cere din nou.

```
int n;
do {
  printf("Introduceti un numar intre 1 si 10");
  scanf("%i", &n);
} while ( (n<1) || (n >10) );
```

Instructiunea WHILE

Instrucţiunea de ciclare while

- execuţă o instrucţiune **cât timp** o expresie are valoare nenulă (adevărată)
- Sintaxa


```
while(expresie)
instructiune
```

- efect:
  - dacă valoarea expresie (conditie) este 0, atunci se opreşte execuţia lui while,
  - altfel se execută instructiune şi se repetă while încă o dată.
- Instructiunea **instr** poate fi o instrucțiune complexă (e.g. un bloc de instrucțiuni)

Instrucţiunea de ciclare while (cont)

- Să se calculeze suma numerelor de la 0 la n.

Instrucţiunea de ciclare while (cont)

- Să se calculeze suma numerelor de la 0 la n.

```
int i=0,s=0;
while(i<n) {
 s=s+i;
 i=i+1;
}
```

Ce afiseaza codul urmator?

```
int  x = 3;
int  i = 0;
while (i < 3) {
  x = x + 1;
  i = i+ 1;
}
printf (“x = %d”, x);
printf (“i = %d”, i);
```

Instrucţiunea de ciclare while (cont)

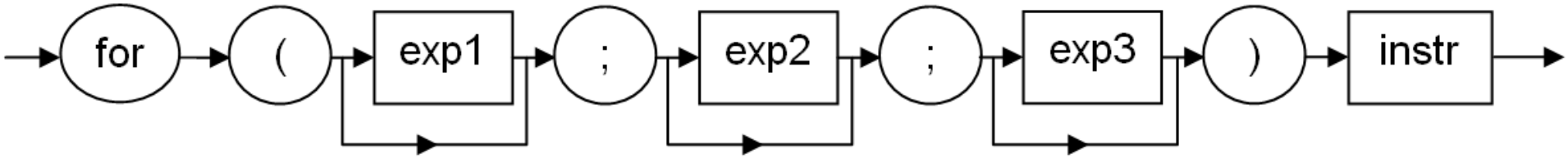
- Se dorește calcularea sumei cifrelor unui număr intreg.

```
int s = 0; //initial suma este 0
   while(n != 0) {
      int c = n%10;//citim ultima cifra
     //se aduna la s ultima cifra
     s = s + c;
     //se scoate ultima cifra din numar
     n=n/10;
   }
   printf("suma cifrelor este %i\n", s);
```

Instructiunea FOR

Instrucţiunea de ciclare for

- execută o instrucţiune cât timp o expresie are valoare nenulă
- sintaxa:




```
for(exp1; exp2; exp3)
    instructiune;
```

- efect:
  - se evaluează exp1 (**initializare**) o singură dată.
  - dacă exp2 (**continuare**) este 0, se opreşte for-ul.
  - Altfel, se execută instr, apoi exp3 (**actualizare**) şi se repetă for-ul (fără iniţializare).

Instrucţiunea de ciclare for (cont)

- Să se afiseze numerele impare mai mici decat n.

Instrucţiunea de ciclare for (cont)

- Să se afiseze numerele impare mai mici decat n.

```
int i;
for(i = 1; i <= n; i += 2)
   printf("%i ", i);
```

Instrucţiunea de ciclare for (cont)

- Să se calculeze produsul primelor n
- numere naturale (n!).

Instrucţiunea de ciclare for (cont)

- Să se calculeze produsul primelor n
- numere naturale (n!).

```
/* declaram p–produsul numerelor pana la
iteratia curenta, i–iteratia (numarul) curent*/

int p = 1, i;
for(i = 1; i <= n; ++i)
// la fiecare pas, produsul curent se
// inmulteste cu numarul curent
     p *= i;
printf("%i ! este %i \n", n, p);
```

Exercitiu

- Scrieti un program care afiseaza numerele divizibile cu 3 mai mici decat un numar n, citit de la tastatura.

Solutia 1

```
#include<stdio.h>

int main(){
   int n;
   printf(”Introduceti limita”);
   scanf(”%d”, &n);
   for(i = 0; i <= n; i += 3)
     printf(”%d ”, i);

   return 0;
 }
```

Solutia 2

```
#include<stdio.h>

int main(){
   int n;
   printf(”Introduceti limita”);
   scanf(”%d”, &n);
   for(i = 0; i <= n; i += 1)
         if(i%3==0)
            printf(”%d ”, i);

   return 0;
 }
```

Ce afiseaza codul urmator?

```
int x = 8;
do{
    printf("%d\n", x);
}while(x > 0);
```

```
int x = 8;
do{
    printf("%o\n", x);
    x--;
}while(x > 0);
```

Ce se afiseaza la executia codului urmator?

```
int i;
for(i=0; i<100;){
   printf("%d ", i);
   i = i+1;
}

/*************************/

 for(;;) {
         printf("%d ",10);
    }
```

```
int i;
for(i=0;i<=5;i++);
    printf("%d",i)


/*********************************/

int i;
for(i=0;i<=5;i++)
    printf("%d",i)
```

Exercitiu

- Afisati numerele de forma 5k+3 mai mici decat un numar n, citit de la tastatura.

Instructiuni de salt

Instrucţiunea de salt break

- opreşte execuţia unei unei instrucţiuni switch, for, while sau do-while şi determina continuarea execuţiei programului cu prima instrucţiune după aceasta.
- Sintaxa: break;

Scrieti un program care afla cel mai mic numar natural n, pentru care suma numerelor de la 1 la n este cel putin 12 (sau 100, sau orice alta valoare). (folosind for, while si do-while)

Instrucţiunea de salt break (cont)

```
#include<stdio.h>

void main(){
  int s=0, i=0;
  for (; ; i++){
       s+=i;
       if(s>=100) break;
  }
  printf("Suma numerelor de la 1 la %d este %d",
  i, s);
}
```

Instrucţiunea de salt break (cont)

- Se dorește afișarea numerelor de la a la b, **până la** prima valoare care este divizibilă cu 7.

Instrucţiunea de salt break (cont)

- Se dorește afișarea numerelor de la a la b, **până la** prima valoare care este divizibilă cu 7.

```
for(i = a; i <= b; ++i) {
   if (i % 7 == 0)
          break;
    printf(“valoarea lui i este %i\n”, i);
}
//Obs. iterațiile i+1,i+2,…,b nu se mai
execută

printf(“De aici se continua executia daca i se
divide cu 7\n”);
```

Instrucţiunea de salt
continue

- continue opreşte execuţia unei unei iteraţii for, while sau do-while şi reia execuţia instrucţiunii respective de la **iterația următoare**

Problema: se dorește afișarea numerelor de la 1 la 10, mai puțin a celor care sunt divizibile cu 7

Instrucţiunea de salt
continue

Problema: se dorește afișarea numerelor de la 1 la 10, mai puțin a celor care sunt divizibile cu 7

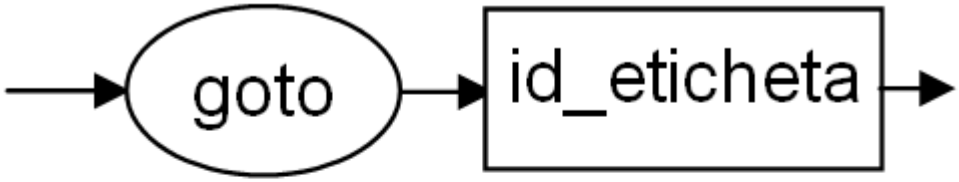
```
for(i = 1; i < 10; ++i) {

  if (i % 7 == 0)
       continue;
  printf(“valoarea lui i este %i\n”, i);

}
```

Instrucţiunea goto

- determină transferul execuţiei programului la prima instrucţiune după eticheta `eticheta`.



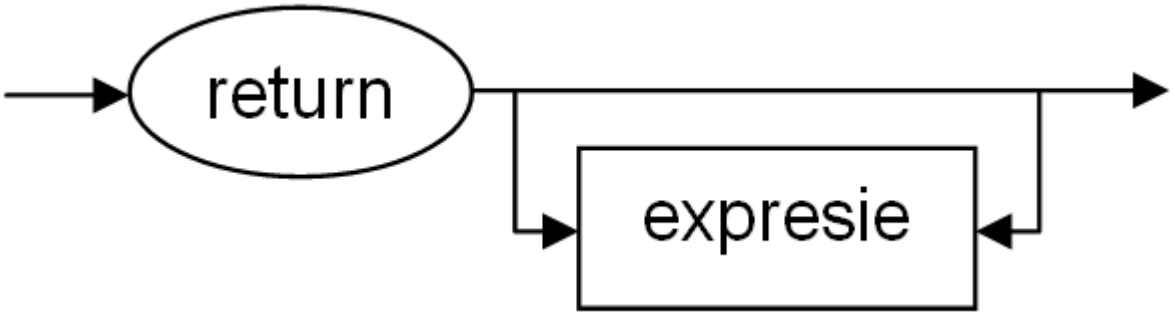
- O etichetă = un identificator unic (în interiorul funcţiei în care este declarat) asociat unei instrucţiuni şi care are semnificaţie doar pentru instrucţiunea `goto`

Instrucţiunea goto (cont)

```
int i = 1;
int suma = 0;
eticheta1:
  if(i<=n){
  suma = suma +i;
  i++;
  goto eticheta1;
  }
```

Instrucțiunea return

- Pentru moment, return este instructiunea care permite iesirea din functia main, eventual cu trimiterea valorii unei expresii catre sistemul de operare



- Sintaxa

return;
return expresie;

Intrebari

1. What is (are) the difference(s) between while and do-while?

2. How do you write an infinite loop using the `while` statement?

3. How do you write an infinite loop using the `for` statement?

Operatori

Expresii in C

O expresie este o secvenţă
- operanzi
  - constante, variabile sau alte expresii.
- operatori (+, -, *, etc)

Exemplu: a = (2*b-c)/(a-b);

Orice expresie are o **valoare** şi un **tip**.
Valoarea şi tipul depind de **operatorii** şi **operanzii** care formează expresia

Exemple de expresii

- printf(“Aceasta este o expresie”);
  - expresie care foloseşte o funcţie ca să afişeze un mesaj
- a = 2 * b;
  - expresie care foloseşte operatori ca să calculeze un rezultat
- a++;
  - expresie care creşte valoarea variabilei a cu 1

Operanzi

- **operand** = informaţie care trebuie prelucrată
  - constante: 10, 5, 'a', "Test“
  - constante simbolice: M_PI, M_E
  - identificator de variabile: x, v[5]
  - numele unei funcţii: sin
  - rezultatul unei funcţii: sin(M_PI)
  - sau o expresie.

Conversii de tip implicite

- tipul unei expresii se poate schimba automat în alt tip pentru a evita pierderi de informaţii
- expresiile care au tip mai "mic" sunt transformate în expresii cu tip mai "mare", dar cu aceeaşi valoare (upcast implicit)
- Exemple

```
    int a=2;
 double b = 2.5;
 double s=a+b;
 // s va avea valoarea 4.5; a este convertit la double


int a = 0;
   char b = '0';
   int dif = a – b;
```

b este convertit la int, codul ASCII al lui '0' este 48 deci dif este -48

Conversii de tip implicite

- int i;
  i = 2.9 ; // i va fi convertit implicit la int, deci va fi 2
  // (cu warning, dar nu cu eroare!)
  - i = ‘a’ ; // i este intreg, deci va lua valoarea codului ASCII al lui ‘a’, deci 97
  - 2 + 3.5 -> 5.5 (tipul double)
    - 3.5 are tipul double, deci 2, desi este de tip intreg, este transformat in dpuble
  - 2 + 3.5f -> 5.5 (tipul float)
    - 2 are tipul întreg, dar este transformat în float
  - 2.1f + 2.2 -> 4.3 (tipul double)
    - 2.1f are tipul float, dar este transformat în double

Conversii de tip explicite

- **Operatorul de conversie (cast)** poate fi folosit pentru a schimba forţat tipul unei expresii.
- Are prioritate mare
- Sintaxa:

```
(tip) expresie
```

  - în faţa expresiei se scrie tipul dorit între paranteze rotunde: (int), (float), etc.
- Exemplu:
  - int a = 5, b = 2;
  - float c, d;
  - c = a/b; // c este 2.0 (împărţire de întregi)
  - d = (float)a/b; // c este 2.5

Reguli

- Regulile de **asociativitate** şi **precedenţă** a operatorilor
  - aceleaşi ca cele din matematică cu unele excepţii
  - **paranteze rotunde ( )** – pentru schimbarea priorităţii de evaluare a subexpresiilor
- Exemplu:
  - a = (a*b-c/d)*(a-b);
  - Pt. a=2, b=3, c=6, d=3, expresia are valoarea -4

Valori logice de adevăr

- În limbajul C, o expresie are valoarea logică de adevăr ***adevărat*** dacă are valoare **nenulă** şi are valoarea logică de adevăr ***fals*** dacă are valoare **nulă**.
- Exemple
  - Expresia 2+3 are valoarea logică de adevăr *adevărat.*
  - Expresia 2-2 are valoarea logică de adevăr *fals*.
  - Expresia x<2 este evaluată de C la 0 sau 1, în funcție de valoarea lui x.

Operatori

- **operator** = **simbol** care determină efectuarea unei operaţii
- După **aritate** = numărul de operanzi cu care lucrează operatorul
  - Unari
  - Binari
  - Ternari: operatorul condiţional ?:
- După **ordinea de succedare** a operatorilor si operanzilor
  - Prefixaţi: ex. -a; !b, --a; ++a
  - Infixaţi: ex. a+b, a%b, a<b
  - Postfixaţi: ex. a++; a--;
- După tipul operanzilor şi al **prelucrării**
  - Aritmetici
  - Relaţionali
  - Logici
  - Pe biţi

Operators (grouped by precedence)

struct member operator	*name*`.`*member*
struct member through pointer	*pointer*`->`*member*
increment, decrement	`++`, `--`
plus, minus, logical not, bitwise not	`+`, `-`, `!`, `~`
indirection via pointer, address of object	`*`*pointer*, `&`*name*
cast expression to type	(*type*) *expr*
size of an object	`sizeof`
multiply, divide, modulus (remainder)	`*`, `/`, `%`
add, subtract	`+`, `-`
left, right shift [bit ops]	`<<`, `>>`
relational comparisons	`>`, `>=`, `<`, `<=`
equality comparisons	`==`, `!=`
and [bit op]	`&`
exclusive or [bit op]	`^`
or (inclusive) [bit op]	`|`
logical and	`&&`
logical or	`||`
conditional expression	$expr_1$? $expr_2$: $expr_3$
assignment operators	`+=`, `-=`, `*=`, ...
expression evaluation separator	`,`

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

Operatori (cont)

- **Precedenţă (prioritate)** = importanţa operatorului într-o expresie
  - operatorii se execută de la prioritate mare la mică
  - operatorii care au aceeaşi prioritate se execută în ordinea în care apar în expresie
- Reguli de asociativitate
  - ordinea aplicarii operatorilor consecutivi de aceeasi precedenta (prioritate).
- Tabela completa cu informatii despre operatori

Operatorul **sizeof**

- Determina dimensiunea in octeti a unui tip de date sau a unei variabile/expresii
- Sintaxa: sizeof(tip) sau sizeof(expresie)
- Exemple:

```
printf(“%d”, sizeof(‘a’)); // 1
sizeof(int) // 4
Sizeof(2.5+2) // 8 //expresia este de tip double
```

Verificati, ca tema pt acasa

```
#include<stdio.h>
int main(){
    printf("%d\t",sizeof(6.5));
    printf("%d\t",sizeof(90000));
    printf("%d",sizeof('A'));
      return 0;
}
```

Operatori aritmetici

- In ordinea descrescatoare a prioritatii:
  - Operatori unari de păstrare/schimbare a semnului: + şi –
  - Operatori binari multiplicativi *, / şi %
  - Operatori binari aditivi + şi –
- Exemplu:
  - int i = –2 + 3 * 4 – 5
  - Este diferit de –(2 + 3 * 4 – 5)
  - Si este diferit de – 2 + 3*(4 – 5)

Operatorul de atribuire

- Sintaxa: `variabila = expresie`
- Operator binar, de prioritate scazuta
- **Asociativitate de la dreapta la stanga**
  - Exemplu: `a = 2 + 3; b = a;`
  - Echivalent cu `b = (a = (2+3));`
  - Echivalent cu `b = a = 2+3;`
- Expresia este evaluata, rezultatul ei este memorat in variabila si este furnizat ca **efect secundar**, asa ca poate fi utilizat mai departe:

```
int a, b, c;
a = b = c = 0;
//echiv cu (a = (b = (c = 0)));
```

Operatori de atribuire compusi

- Expresiile de tip

```
variabila = variabila operator expresie
```

- Se pot scrie prescurtat in forma

```
variabila operator= expresie
```

- Se definesc astfel operatorii:
  - += -= *= /= %=
  - Distinct, vorbim de
  - &= |= ^= <<= >>=
    - sunt operatori pe biti

Exemplu

- int i=3, j=9, k=11;
- i += j;
  - echivalent cu i=i+j;
- k %= j -= 4;
  - asocierea fiind de la dreapta la stanga, deci echivalent cu (k %= (j -= 4));
  - prima data se calculeaza j -= 4,
  - deci j va fi 5, valoare care este folosita prin efect secundar pentru k%=5, deci k va fi 1

Operatori de incrementare (++) si decrementare (--)

- Operanzi: variabile intregi
- prefixati
  - ++variabila
    - Echivalent cu variabila = variabila+1
  - --variabila
    - Echivalent cu variabila = variabila – 1
  - Efect lateral: furnizeaza valoarea variabilei **dupa** incrementare
- postfixati
  - variabila++
  - variabila--
  - Efect lateral: furnizeaza valoarea variabilei **inainte** de incrementare

```
int a = 5;
a++;
printf("a este %d \n", a); // a este  6
int b = a++;
/* il seteaza pe b la valoarea lui a,
   apoi incrementeaza pe a*/
printf("a este %d si b este %d \n",
   a, b); // a este 7 si b este 6

b = ++a;
/*il incrementeaza pe a si apoi
   seteaza pe b la noua valoare a lui
   a*/
printf("a este %d si b este %d \n",
   a, b);
// a este 8 si b este 8
```

Operatori de incrementare (++) si decrementare (--) (cont)

```
int i=3, j=9, K;
K = ++i + j--;
printf(“i este %d, j este %d, k este %d.\n”, i, j, k);
```

Explicatii

- ++ si –- au prioritatea mai mare ca + (binar)
- Expresia este echivalenta cu K = (++i) + (j--);
- ++i are doua efecte:
  - creste pe i, deci i devine 4;
  - expresia ++i se evalueaza, ca efect lateral la valoarea noua a lui i, 4;
- j-- are doua efecte:
  - Scade pe j, deci j devine 8;
  - se evalueaza, ca efect lateral la valoarea veche 9
- k va fi 13 (4+9)

Ce se afiseaza?

```
#include<stdio.h>
int main(){
    int i=1;
    i=2+2*i++;
    printf("%d",i);
    return 0;
}
```

```
#include<stdio.h>
int main(){
    int a=2,b=7,c=10;
    c=a==b;
    printf("%d",c);
    return 0;
}
```

Ce se afiseaza?

```
#include<stdio.h>
int main(){
    int a=0,b=10;
    if(a=0){
        printf("true");
    }
    else{
        printf("false");
    }
    return 0;
}
```

```
#include<stdio.h>
void main(){
    int x;
    x=10, 20, 30;
    printf("%d",x);
    return 0;
}
```

Operatori relationali

- Operatori binari, folositi in expresii ce sunt evaluate la fals (0) sau adevarat (1)
- < <= > >=
- == !=
- Orice valoare **diferita de 0** este interpretata ca **adevarat** in C.

Operatori relationali (cont)

- int a;
- Observatie: faceti diferenta intre conditia

  a==0
- si

  a=0
- Prima testeaza pe a si rezultatul evaluarii este 1 (true) sau 0 (false), in functie de valoarea lui a
- A doua seteaza pe a la 0 si rezultatul evaluarii ei este valoarea din dreapta, adica 0, deci fals.

Operatori logici

- Lucreaza cu operanzi intregi, interpretati ca valori logice
  - ! negatie logica
  - && si logic
  - || sau logic
- ! Este operator unar are prioritatea cea mai ridicata

X	!X
≠0	0
0	1

Operatori logici (cont)

- && si || sunt operatori binari, au prioritatea mai mica decat operatorii relationali

X	Y	X && Y
0	0	0
0	≠0	0
≠0	0	0
≠0	≠0	1

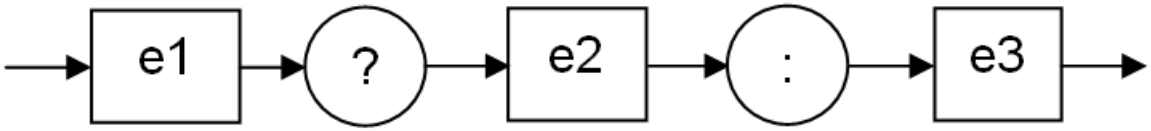
X	Y	X || Y
0	0	0
0	≠0	1
≠0	0	1
≠0	≠0	1

Operatori logici - observații

- Expresiile logice in C se calculeaza prin **scurtcircuitare**
  - daca primul operand al expresiei in care apare operatorul && este 0, sigur rezultatul final este 0, indiferent de valoarea celui de-al doilea.
  - daca primul operand al expresiei in care apare operatorul || este !=0, sigur rezultatul final este 1, indiferent de valoarea celui de-al doilea
  - Deci: dacă primul operand are valorile de mai sus, corespunzator operandului && sau ||, cel de-al doilea operand nu se mai evalueaza.

Operatorul conditional ?:

- operator ternar
  `expr0 ? expr1 : expr2`



- Daca valoarea `e1` este adevarata (`!=0`), se evalueaza `e2`, altfel `e3`
- Tipul si rezultatul expresiei evaluate fiind tipul si rezultatul final al expresiei conditionale.
- Exemplu:

```
max=(a>b)?a:b;
```

- Din punct de vedere al rezultatului final, este echivalent:

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

Exercitiu

- Ce valoare va fi afişată pe ecran?
- int a=3, b=5;
  printf (a==b ? "egale" : "diferite");
- int a=3, b=7%4;
  printf(a==b ? "egale" : "diferite");
- int a=3, b=6;
  printf("a si b sunt %s", a==b ? "egale" :
  "diferite");
- int a; a = 3,5,6;
  printf("a = %d", a);

Operatori pe biti

Operatori pe biţi

- se aplică **fiecărui bit** din reprezentarea operanzilor **intregi**
- spre deosebire de restul operatorilor care se aplică *valorilor* operanzilor.
- ~ complementariere
  - transforma fiecare bit din reprezentarea operandului in complementarul sau -- bitii 1 in 0 si cei 0 in 1.
- & si
- ^ sau exclusiv
- | sau
  - realizeaza operatiile si, sau exclusiv, respectiv sau intre toate perechile de biti corespunzatori ai operanzilor.

Operatori pe biti (cont)

a	b	a&b	a^b	a|b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	1

Din tabela de mai sus se observa ca aplicand unui bit:

operatorul & cu 0, bitul este resetat (sters)
operatorul & cu 1, bitul este neschimbat
operatorul | cu 1, bitul este setat
operatorul | cu 0, bitul este neschimbat
operatorul ^ cu 1, bitul este complementat.
operatorul ^ cu 0, bitul este neschimbat.

Operatori pe biti (cont)

- >> deplasare la dreapta
- << deplasare la stanga
- primul operand e cel ai carui biti sunt deplasati,
- al doilea indica numarul de biti cu care se face deplasarea
  - a>>n este echivalent cu $a/2^n$
  - a<<n este echivalent cu $a*2^n$
- Este indicat a se realiza inmultirile si impartirile cu puteri ale lui 2 prin deplasari; deplasarile se executa intr-un timp mult mai scurt.

Operatori pe biti (cont)

- Exemple
- int i = 256;
- i *= 8; // echivalent cu i<<3;
- i /= 4; // echivalent cu i>>2;

Operatori pe biti (cont)

```
#include <stdio.h>

main() {

   unsigned int a = 60;    /* 60 = 0011 1100 */
   unsigned int b = 13;   /* 13 = 0000 1101 */
   int c = 0;

   c = a & b;          /* 12 = 0000 1100 */
   printf("Line 1 - Value of c is %d\n", c );

   c = a | b;          /* 61 = 0011 1101 */
   printf("Line 2 - Value of c is %d\n", c );

   c = a ^ b;          /* 49 = 0011 0001 */
   printf("Line 3 - Value of c is %d\n", c );
```

```
a = 60;    /* 60 = 0011 1100 */
b = 13;    /* 13 = 0000 1101 */


a = 60;    /* 60 = 0011 1100 */
c = ~a;        /*-61 = 1100 0011 */
   printf("Line 4 - Value of c is %d\n", c );

a = 60;    /* 60 = 0011 1100 */
c = a << 2; /* 240 = 1111 0000 */
   printf("Line 5 - Value of c is %d\n", c );

a = 60;    /* 60 = 0011 1100 */
 c = a >> 2; /* 15 = 0000 1111 */
   printf("Line 6 - Value of c is %d\n", c );
}
```

Tema pentru acasa

- Care sunt tipurile şi valorile expresiilor?

```
int a = 5, b = 2;
float c = 1;
9 - a
9/a
9.0/a
!a
b<<2
a>>1
!(a-(1<<b)-1)
a-1<<b-1 > 5
```

Tema pentru acasa

- Ce va afisa programul de mai jos?

```
#include <stdio.h>
int main (void) {
   double x, y;
   int a = 7, b = 2;
   x = (int)1.5 * (double)(a/b);
    x = (int)1.5 * ((double)a/b);
   printf(“x=%lf, y=%lf\n", x, y);
}
```