

Programarea calculatoarelor

Limbajul C



CURS 9



Operatori pe biți

Structuri, Uniuni, Enumerări



Operatorii pe biți

- Se aplică fiecărui bit din reprezentarea *operanzilor întregi*
- Restul operatorilor se aplică valorilor operanzilor

- & și
- | sau
- ^ XOR (sau exclusiv)
- ~ complement față de 1
- << deplasare la stânga
- >> deplasare la dreapta

Exemplu:

- Exemplu:

7 00000000000000111

8 0000000000001000

7 & 8 0000000000000000

7 & 8 = 0

7 && 8 = 1 (adevărat și adevărat → adevărat).

- Operatorul ~ transformă fiecare bit din reprezentarea operandului în complementarul său (biții 1 în 0 și cei 0 în 1)

Operatorii de deplasare

- Primul operand este cel al cărui biți sunt deplasați
- Al doilea indică numărul de biți cu care se face deplasarea:
$$a \ll n, a \gg n$$
- La *deplasarea la stânga* cu o poziție, bitul cel mai semnificativ se pierde, iar în dreapta se completează cu bitul 0.
- La *deplasarea la dreapta* cu o poziție, bitul cel mai puțin semnificativ se pierde, iar în stânga se completează cu un bit identic cu cel de semn.
- Cu excepția cazurilor când se produce depășire, deplasarea la stânga cu n biți echivalează cu înmulțirea cu 2 la puterea n .
- Deplasarea la dreapta cu n biți echivalează cu împărțirea cu 2 la puterea n .

Observație

- Este indicat să se realizeze înmulțirile și împărțirile cu puteri ale lui 2 prin deplasări (necesită un timp mult mai scurt):

int i;

- $i * = 8;$ ~ $i << = 3;$
- $i / = 4;$ ~ $i >> = 2;$
- $i * = 10;$ ~ $i = i << 3 + i << 1;$

Exemple

char x	x dupa fiecare executie	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

Exemplu

- Se consideră n , p întregi. Să se seteze pe 1 bitul p din reprezentarea lui n .

```
#include<stdio.h>
{
    unsigned int n=5,p=1;
    n = n | (1<<p);
    printf(“%u\n”n);
}
```

Exemplu

- Să se scrie o secvență care construiește o mască ce conține n biți de 1 începând cu poziția p, spre dreapta (bitul cel mai puțin semnificativ se consideră ca fiind de poziție 0)
- ```
int masca=0, unu=1;
int i, p=5, n=3;
unu << = p; // bitul 1 este pe pozitia p
for (i=1; i<=n; i++) {
 masca | = unu;
 unu >> = 1;
}
```





# Exerciții

---

1. Se consideră  $n$  și  $p$  întregi. Să se seteze pe 0 bitul  $p$  din reprezentarea lui  $n$ .
2. Să se afișeze un număr întreg fără semn în binar folosind extragerea selectivă de biți cu o mască ce conține un singur bit 1.

# Rezolvare 1

---

```
#include<stdio.h>
void main()
{
 unsigned int n=7,p=1;
 n&= ~(1<<p);
 printf(“%u\n”n);
}
```

# Rezolvare 2

---

```
#include<stdio.h>
void main () {
 int n;
 printf("n= "); scanf("%d",&n);
 unsigned int mask = 1<<31; /* bitul 31 =1, ceilalti zero*/
 while (mask) {
 printf ("%d", (mask & n) ? 1: 0);
 mask >>= 1; /* deplasare masca la dreapta cu un bit*/
 }
 printf("\n");
}
```

# Structură

---

- O structura este un tip de date care ne permite gruparea unor elemente eterogene.
- Este o colecție de una sau mai multe variabile (câmpuri), grupate sub un singur nume.
- Printr-o declarație *struct* se definește un nou tip de date de către utilizator.

- Definiere C:

```
struct nume_structura {
 tip nume_camp_1;
 tip nume-camp_2;
 ...
} [lista_variabile_structura];
```

# Exemple

---

- structura moment de timp

```
struct time {
 int ora, min, sec;
};
```

- structura activitate

```
struct activ {
 char numeact[30]; // nume activitate
 struct time start; // ora de incepere
 struct time stop; // ora de terminare
};
```

# Declarare variabile de tip structură

---

- Declararea unor variabile de un tip structură se poate face fie după declararea tipului structură, fie simultan cu declararea tipului structură.
- Exemple:
- `struct time t1, t2, t [100]; // t este vector de structuri`
- `struct complex {  
    float re, im;  
} c1,c2,c3;  
struct complex cv[200];  
    // un vector de numere complexe`

# Initializare și referire elemente structură

---

- struct data\_calendaristica {  
    int zi;  
    int luna;  
    int an;  
} d;

- inițializare :

```
struct data_calendaristica azi = { 5, 4, 2006 };
```

```
struct complex c1= {1,-1}, c2= {2,3};
```

- Pentru referire se utilizează operatorul “.”

d.zi, d.luna, d.an

# Observații

---

- In structuri diferite pot exista câmpuri cu același nume, dar într-o aceeași structură numele de câmpuri trebuie să fie diferite.
- Definirea tipului structură (cu sau fără *typedef*) se face la începutul fișierului sursă care conține funcțiile (înaintea primei funcții)
- Câmpurile unei variabile structură nu se pot folosi decât dacă numele câmpului este precedat de numele variabilei structură din care face parte, deoarece există un câmp cu același nume în toate variabilele de un același tip structură!
- Nu pot spune *zi* ci *d.zi* !!



# Observații

---

- Dacă un câmp este la rândul lui o structură, atunci numele unui câmp poate conține mai multe puncte ce separă numele variabilei și câmpurile de care aparține (în ordine ierarhică)!

- Exemplu:

```
struct activ a;
```

```
printf ("%s începe la %d: %d și se termina la %d: %d
\n", a.numeact, a.start.ora, a.start.min, a.stop.ora,
a.stop.min);
```

# Asocierea de sinonime pentru tipuri structuri

---

- Exemplu fără asociere sinonim:

```
struct Carte
{
 int val;
 char cul[20];
};
struct Carte c1, c2;
```

- Observație: într-un fișier CPP se poate utiliza și fără a folosi cuvântul cheie *struct*!

```
Carte c1, c2
```

# Asocierea de sinonime pentru tipuri structuri

---

- Se utilizează *typedef*
- Permite atribuirea unui nume oricărui tip
- Numele se poate folosi apoi la fel cu numele tipurilor predefinite ale limbajului.
- Sintaxa declarației *typedef* este la fel cu sintaxa unei declarații de variabilă, dar se declară un nume de tip și nu un nume de variabilă!

# Exemplu

---

```
typedef struct card
{
 int val;
 char cul[20];
} Carte;
Carte c1, c2;
c1.val = 10;
strcpy (c1. Cul, "caro");
c2 = c1;
```

- Cu typedef structura poate fi și anonimă (poate lipsi cuvântul card)!

# Asocierea de sinonime pentru tipuri structuri

---

- definire nume tip simultan cu definire tip structură:

```
typedef struct {
 float re, im;
} complex;
```

- definire nume tip după definire tip structură:

```
typedef struct activ activitate;
```

# Observație

---

- Se pot folosi ambele nume ale unui tip structură (cel precedat de *struct* și cel dat prin *typedef*), care pot fi chiar identice:
- ```
typedef struct complex {  
    float re; float im;  
} Complex;  
Complex c1;  
struct complex c2, c3;
```
- ```
typedef struct point {
 double x, y;
} point;
struct point p[100];
```

# Utilizarea tipurilor structură

---

- Un tip structură poate fi folosit în :
  - declararea de variabile structuri sau pointeri la structuri
  - declararea unor argumente formale de funcții (structuri sau pointeri la structuri)
  - declararea unor funcții cu rezultat de tip structură
- Operațiile posibile cu variabile de un tip structură sunt:
  - aplicarea operatorilor & și sizeof
  - atribuirea între variabile de același tip structură
    - se folosește operatorul de atribuire “=”
    - nu se pot folosi alți operatori ai limbajului
    - trebuie definite funcții pentru operații cu structuri: comparații, operații aritmetice, operații de citire-scriere etc.
  - transmiterea ca argument efectiv la apelarea unei funcții.
  - transmiterea ca rezultat al unei funcții, într-o instrucțiune *return*.

# Exercitii - Structuri

---

- I Să se definească o structura "time" care grupează 3 întregi ce reprezintă ora, minutul și secunda pentru un moment de timp. Să se scrie funcții pentru:
  1. Verificare corectitudine ora
  2. Citire moment de timp
  3. Scriere moment de timp
  4. Comparare de structuri "time".
- I Program pentru citirea și ordonarea cronologică a unor momente de timp și afișarea listei ordonate, folosind funcțiile anterioare.



# Exemplu: Ordonarea unui vector de structuri "time"

---

```
#include<stdio.h>
typedef struct time {
 int ora, min, sec;
}time;

void wrtime (time t) { // scrie ora,min,sec
 printf ("%02d:%02d:%02d \n", t.ora,t.min,t.sec);
}

int cmptime (time t1, time t2) { // compara momente de timp
 int d;
 d=t1.ora - t2.ora;
 if (d) return d;
 d=t1.min - t2.min; // <0 daca t1<t2 și >0 daca t1>t2
 if (d) return d; // rezultat negativ sau pozitiv
 return t1.sec - t2.sec; // rezultat <0 sau =0 sau > 0
}
```

# Exemplu - continuare

---

```
int corect (time t) { // verifica daca timp plauzibil
 if (t.ora < 0 || t.ora > 23) return 0;
 if (t.min < 0 || t.min > 59) return 0;
 if (t.sec < 0 || t.sec > 59) return 0;
 return 1; // plauzibil corect
}

time rdtime () { // citire ora
 time t;
 do {
 scanf ("%d%d%d", &t.ora, &t.min,&t.sec);
 if (! corect (t))
 printf ("Date gresite, repetati introducerea: \n");
 else break;
 } while (1);
 return t;
}
```

# Exemplu - continuare

---

```
void sort (time a[], int n) { // ordonare vector de date
 int i,gata; time aux;
 do {
 gata=1;
 for (i=0;i<n-1;i++)
 if (cmptime(a[i],a[i+1]) > 0) {
 aux=a[i]; a[i]=a[i+1]; a[i+1]=aux; gata=0;
 }
 } while (! gata);
}
int main () {
 time t[30];
 int n,i;
 printf("introducere n: "); scanf("%d",&n);
 for(i=0;i<n;i++) t[i] = rdtime();
 sort (t, n);
 for (i=0 ;i<n ;i++) wrtime(t[i]);
 return 1;
}
```

# Avantajele utilizării tipurilor structură

---

- Programele devin mai explicite dacă se folosesc structuri în locul unor variabile separate.
- Se pot defini tipuri de date specifice aplicației iar programul reflectă mai bine universul aplicației.
- Se poate reduce numărul de argumente al unor funcții prin gruparea lor în argumente de tipuri structură și deci se simplifică utilizarea acelor funcții.
- Se pot utiliza structuri de date extensibile, formate din variabile structură alocate dinamic și legate între ele prin pointeri (liste înlănțuite, arbori s.a).

# Funcții cu rezultat de tip structură

---

- O funcție care produce un rezultat de un tip structură poate fi scrisă în două moduri:

1. Funcția are rezultat de tip structură:

```
// citire număr complex (varianta 1)
complex readx () {
 complex c;
 scanf ("%f%f ", &c.re, &c.im);
 return c;
}
// utilizare
complex a[100];
...
for (i=0;i<n;i++) a[i]=readx();
```

# Funcții cu rezultat de tip structură

---

2. Funcția depune rezultatul la adresa primită ca argument (pointer la tip structură, pentru a putea modifica):

```
// citire număr complex (varianta 2)
void readx (complex * px) { // pointer la o structură complex
 scanf ("%f%f ", &(*px).re, &(*px).im);
// sau scanf ("%f%f ", &px->re, &px->im);
}
// utilizare
complex a[100];
...
for (i=0;i<n;i++) readx (&a[i]);
// adresa variabilei structură a[i]
```

# Observații

---

- Notația `px→re` este echivalentă cu notația `(*px).re`
  - Câmpul “re” al structurii de la adresa `px`
- Pentru structurile care ocupă un număr mare de octeți este mai eficient să se transmită ca argument la funcții adresa structurii (un pointer) în loc să se copieze conținutul structurii la fiecare apel de funcție și să se ocupe loc în stiva de variabile *auto*, chiar dacă funcția nu face nici o modificare în structura a cărei adresă o primește!

# Uniunea (reuniunea) - union

---

- Definește un grup de variabile care nu se memorează simultan ci alternativ.
- Se pot memora diverse tipuri de date la o aceeași adresă de memorie.
- Alocarea de memorie se face (de către compilator) în funcție de variabila ce necesită maxim de memorie.
- O uniune face parte de obicei dintr-o structură care mai conține și un câmp discriminant, care specifică tipul datelor memorate (alternativa selectată la un moment dat).



# Structuri predefinite

---

- “struct tm” definită în <time.h>
- are componente ce definesc complet un moment de timp:

```
struct tm {
 int tm_sec, tm_min, tm_hour; // secunda, minut, ora
 int tm_mday, tm_mon, tm_year; // zi, luna, an
 int tm_wday, tm_yday; // nr zi în saptamana și în an
 int tm_isdst; // 1=se modifica ora (iarna/vara), 0= nu
};
```

- Există funcții ce lucrează cu această structură:  
asctime, localtime, etc

# Cum se poate afișa ora și ziua curentă, folosind numai funcții standard

---

```
#include <stdio.h>
#include <time.h>
int main(void) {
 time_t t; // time_t este alt nume pentru long
 struct tm *area; // pentru rezultat funcție localtime
 t = time (NULL); // obtine ora curenta
 area = localtime(&t); // conversie din time_t în struct tm
 printf ("Local time is: %s", asctime(area));
}
```

Observație:

- `asctime( struct tm* t)` returnează un șir ce reprezintă ziua și ora din structura `t`. Șirul are următorul format: **DDD MMM dd hh:mm:ss YYYY**
- funcția “`time`” transmite rezultatul și prin numele funcției și prin argument: `long time (long*)`, deci se putea apela și: `time (&t)`;

# Structuri predefinite

---

- “struct stat” definită în fișierul <sys/stat.h>
- reunește date despre un fișier, cu excepția numelui

```
struct stat {
 short unix [7]; // fără semnificatie în sisteme Windows
 long st_size; // dimensiune fisier (octeti)
 long st_atime, st_mtime;
 // ultimul acces / ultima modificare
 long st_ctime; // data de creare
};
```

- Funcția “stat” completează o astfel de structură pentru un fișier cu nume dat:

```
int stat (char* filename, struct stat * p);
```

# Exemplu

---

- Pentru a afla dimensiunea unui fișier normal (care nu este fișier director) vom putea folosi funcția următoare:

```
long filesize (char * filename) {
 struct stat fileattr;
 if (stat (filename, &fileattr) < 0) // daca fisier negasit
 return -1;
 else // fisier gasit
 return (fileattr.st_size);
 // campul st_size contine lungimea
}
```

# Enumerări

---

- Folosite pentru a da nume simbolice unui șir de valori numerice.
- Sintaxa:  
`enum nume_opt { lista_constante } lista_variabile_opt ;`
- Observație: constantele pot avea specificate valori (și o valoare se poate repeta)
- Exemplu:  
`enum luni_curs {ian=1, feb, mar, apr, mai, iun, oct=10, nov, dec};`
- Implicit, șirul valorilor e crescător cu pasul 1, iar prima valoare e 0
- Un nume de constantă nu poate fi folosit în mai multe enumerări

# Enumerări

---

- Tipurile enumerare sunt tipuri întregi
- Variabilele enumerare se pot folosi la fel cu variabilele întregi
- Cod mai lizibil decât prin declararea separată de constante:
- ```
enum {D, L, Ma, Mc, J, V, S} zi;  
/* tip anonim; declară doar var.zi, tipul nu are nume, deci nu  
mai putem declara altundeva variabile */  
sau:  
typedef enum {D, L, Ma, Mc, J, V, S} Zi;  
// tip enumerare cu numele Zi; se pot declara variabile  
int nr_ore_lucru[7]; /* număr de ore pe zi */  
for (zi = L; zi <= V; ++zi) nr_ore_lucru[zi] = 8;
```

Exemplu

- Să se scrie un program care declară o structură pentru un număr generic, folosind o enumerare pentru valoarea numărului și un câmp tip ce va spune ce fel de număr este (întreg – i, întreg lung – l, real –f, real dubla precizie – d).
- Să se scrie o funcție ce citește o variabilă de tipul structurii definite
- Să se scrie o funcție ce afișează valoarea unei variabile de tipul structurii definite
- Pentru câmpul discriminant se poate defini un tip enumerare, împreună cu valorile constante (simbolice) pe care le poate avea. Să se refacă programul!

Rezolvare

```
#include<stdio.h>
// structura pentru un număr de orice tip
typedef struct{
    char tipn;          // tip număr (un caracter)
    union {
        int ival; long lval; float fval; double dval;
    } v; // valoarea numărului
} număr;

// afisare număr
void write (număr n) {
    switch (n.tipn) {
        case 'i': printf ("întreg %d ",n.v.ival);break;
        case 'l': printf ("întreg lung %ld ",n.v.lval);break;
        case 'f': printf ("Real %.5f ",n.v.fval);break;
        case 'd': printf ("Real dublu %.15lf ",n.v.dval);break;
    }
}
```


Rezolvare

```
număr read (char tip) {
    număr n;
    n.tipn=tip;
    switch (tip) {
        case 'i': scanf ("%d", &n.v.ival); break;
        case 'l': scanf ("%ld", &n.v.lval); break;
        case 'f': scanf ("%f", &n.v.fval); break;
        case 'd': scanf ("%lf", &n.v.dval);
    }
    return n;
}

int main () {
    număr a, b, c, d;
    a = read('i'); b=read('l'); c = read('f'); d=read('d');
    write(a); write(b); write(c); write(d);
    getchar();
    return 1;
}
```

Rezolvare 2

```
#include<stdio.h>
enum tnum {I,L,F,D} ; // definire tip "tnum"
typedef struct{
    tnum tipn;          // tip număr (un caracter)
    union {
        int ival; long lval; float fval; double dval;
    } v; // valoarea numărului
} număr;

void write (număr n) {
    switch (n.tipn) {
        case I : printf ("%d ",n.v.ival); break;           // int
        case L: printf ("%ld ",n.v.lval);break;           // long
        case F: printf ("%f ",n.v.fval);break;           // float
        case D: printf ("% .15lf ",n.v.dval);break;       // double
    }
}
```

Rezolvare 2

```
număr read (tnum tip) {  
    număr n;  
    n.tipn=tip;  
    switch (tip) {  
        case l: scanf ("%d", &n.v.ival); break;  
        . . .  
    }  
    return n;  
}
```

```
int main () {  
    număr a,b,c,d;  
    a = read(l);  
    write(a);  
    . . .  
}
```