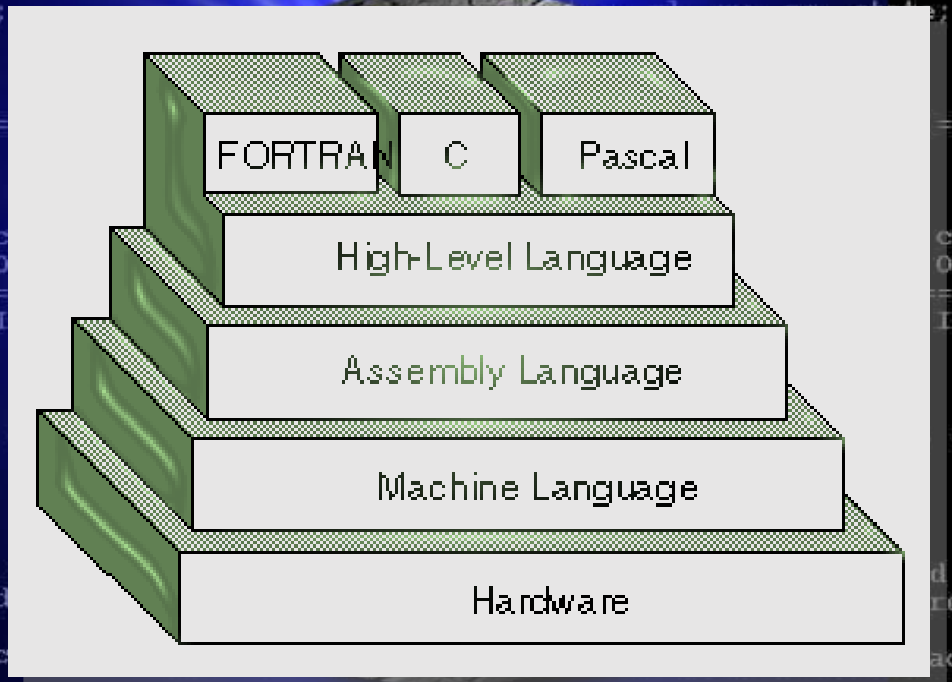
Limbajul C

1. Construcții de bază
2. Tipuri de date
3. Expresii
4. Instrucțiuni
5. Operaţii de intrare/ieşire cu tastatura/monitorul
6. Tipuri dinamice de date
7. Subprograme



God's Programming Language

1. Construcţii de bază

a) Identificatorii
b) Comentariile
c) Expresiile
d) Declaraţiile
e) Instrucţiunile
f) Funcţiile (subprogramele)
g) Programul
h) Directivele de preprocesare

a) Identificatorii

Identificatori = denumiri asociate entităţilor referite în program;

= succesiune de:

- *litere* (mici şi/sau mari, din alfabetul latin)
- *cifre zecimale* (de la 0 la 9)
- _ (liniuţa de subliniere)

REGULĂ: Un identificator nu poate începe VREODATĂ cu o cifră

Exemple:

a, x, X, abc, Abc, x1, x_1, alfa, pret_unitar, student, CodProdus, _gigel

Limbajul C este un limbaj CASE-SENSITIVE

Cuvinte rezervate -> identificatori predefiniţi în limbaj

Exemple:

for, do, if, else, char, long, float, while, return

RECOMANDARE: La alegerea identificatorilor de vor folosi denumiri **SUGESTIVE** relativ la semnificația acestora (exp. minim – min, maxim – max, suma – s, produs – p etc.)

b) Comentariile

Comentarii = secvențe de text compuse din orice caractere admise în setul limbajului și care nu se compilează;

/* …… */

Exemple:

/* Acesta este un comentariu */

LIZIBILITATE a codului sursă

/* Citire date de intrare;
Prelucrare propriu-zisa;
Afisare rezultate */

Observație: în C++ avem în plus comentarii pe o singură linie sursă

// ……..

Exemplu: // initializare variabile

c) Expresiile

Expresii = construcții formate din:

- operanzi (date numerice, logice, de tip caracter etc.)
- operatori (aritmetici, relaționali, logici etc.)

și a căror evaluare produce o valoare de un anumit tip.

<operator_unar><expresie> **-x**

<expresie><operator_binar><expresie> **a+b**

<expresie><operator_ternar><expresie><expresie>

d) Declarații

Declarații = sunt construcții pentru definirea și descrierea datelor

Date:

- constante și/sau variabile (dpdv al numărului de valori luate pe parcursul unei execuții a programului)

- date scalare și/sau structuri de date etc. (tipuri structurate: masiv, articolul – *struct*, *union*, șir de caractere – neimplementat nativ în limbajul C)

e) Instrucţiunile

Instrucțiunile:

- sunt construcții sintactice ale limbajului, terminate de caracterul **;** (punct și virgulă);

a=10 - expresie de atribuire

a=10; - instrucțiune de atribuire

- indică operațiile (acțiunile) care se aplică datelor în vederea obținerii rezultatelor scontate prin algoritm;
- pot fi clasificate în:

  - simple: vidă, de atribuire, apel de procedură, break, goto etc.

  - compuse { instructiune; }

  - structurate = acele instrucțiuni care realizează implementarea structurilor fundamentale din programarea structurată (cele 7) într-un limbaj de programare (*if-else, switch, while, do-while, for*)

f) Funcţiile (subprogramele) - function(s)

Funcţiile = sunt entităţi (**subprograme**) care pot fi proiectate, realizate şi compilate independent, dar care nu se execută decât împreună cu o altă entitate de program, numită apelatoare (de cele mai multe ori aceasta fiind chiar programul principal).

antet

{

corp

}

antet -> tip nume(lista_parametrilor_formali) - se numeşte ***header***

Exemplu: float suma(int n, float v[20])

corp -> conţine o instrucţiune compusă: conţine declaraţiile locale şi instrucţiunile executabile care implementează algoritmul

-> se execută până la executarea ultimei instrucţiuni sau până la executarea instrucţiunii *return* (trebuie folosit OBLIGATORIU în cazul subprogramelor de tip FUNCŢIE – cele care returnează ALTCEVA decât *void*)

Apelul subprogramelor: nume(lista_parametrilor_reali)

g) Programul principal

Programul este:

- construcţia sintactică de cel mai înalt nivel;

- constituit din declarații și instrucțiuni executabile;

- o înșiruire de subprograme și declarații, printre care trebuie să existe o funcție denumită *main*, lansată în execuție la rularea programului
  - main()
  - int main()
  - ***void* main() - RECOMANDARE**
  - ....

h) Directivele de preprocesare

Preprocesarea: se efectuează substituiri asupra textului sursă scris de programator – are loc înainte de compilare (permite includerea conținutului de fișiere – de regulă fișiere *header*, definirea de macrouri și compilare condiționată)

Directive: <directiva>::= **#** <cuvant_rezervat>[<parametri>]

❖Directiva #include este folosită pentru includerea de fișiere cu text (cod) sursă într-un program

#include <specificator_de_fisier> - se caută în toată structura de directoare livrate odată cu limbajul

sau

#include ”specificator_de_fisier” - se caută doar în directorul curent de lucru

Exemplu: #include <stdio.h> //pentru *printf* și *scanf*

❖Directiva #define este folosită pentru a substitui unele secvențe de caractere cu altele

#define nume descriere

unde atât *nume* cât şi *descriere* sunt şiruri de caractere.

1. definirea de *constante simbolice*
2. definirea de *macrodefiniţii*

- **Exemple:**

```
#define PI 3.1415    //constanta simbolica
#define N 10
#define M 10
#define MAX  (M+N)       //macrodefinitie
#define DIM(a,b) (a)*(b)
```

Discuții și exemple: ....

float a=PI+5; // dupa preprocesare codul sursa va fi: float a=3.1415+5;

// dupa compilare variabila a va avea valoarea 8.1415

Macrodefiniţia (macro-ul):

- este un nume simbolic asociat unei secvențe fixe sau variabile (cu o parte fixă și una variabilă) de text sursă;
- este folosită pentru a ușura scrierea textului sursă;
- este un nume acordat unei secvențe care se repetă frecvent, identic sau cu mici variații;
- macro-ul este tratat la *preprocesare*.

Expandarea macrodefiniţiei = operația de substituire a numelui cu descrierea

❖Directiva #undef este folosită pentru dezactivarea substituirii din punctul în care apare directiva #undef până la sfârșitul programului sau până la redefinirea lui *nume*

Exemplu:

#undef nume

2. Tipuri de date

<table>
<tr><th>După modul (momentul) de alocare a memoriei</th><th>După numărul de valori memorate</th><th>Tipuri existente în C</th></tr>
<tr><td rowspan="6">Statice (la compilare)</td><td rowspan="3">Simple</td><td>Întregi</td></tr>
<tr><td>Reale</td></tr>
<tr><td>Caracter</td></tr>
<tr><td rowspan="3">Structurate</td><td>Masiv</td></tr>
<tr><td>Articol</td></tr>
<tr><td>Fişier (extern)</td></tr>
<tr><td rowspan="2">Dinamice (la execuție)</td><td rowspan="2">Simple</td><td>Pointer</td></tr>
<tr><td>Referinţă</td></tr>
</table>

2.1 Tipuri simple de date

a) Tipuri predefinite de date

Grupa de dată	Tipul	Lungime (octeţi)*	Domeniu de valori	Mod de reprezentare
Întreg	unsigned char	1	0..255 (0..2^8-1)	Codul ASCII al caracterului. Poate fi prelucrat ca un caracter sau ca un întreg cu/fără semn.
	[signed] char	1	-128..127 (-2^7..2^7-1)	
	unsigned short [int]	2	0..65535	Virgulă fixă aritmetică
	[signed] short [int]	2	-32768..32767	Virgulă fixă algebrică
	unsigned [int]	2 sau 4	0..65535 sau 0..2^{32}-1	Virgulă fixă aritmetică
	[signed] [int]	2 sau 4	-32768..32767 sau -2^{31}..2^{31}-1	Virgulă fixă algebrică
	unsigned long	4	0..2^{32}-1	Virgulă fixă aritmetică
	[signed] long [int]	4	-2^{31}..2^{31}-1	Virgulă fixă algebrică
Real	float	4	3.4*10^{-38}..3.4*10^{38}	Virgulă mobilă simplă precizie
	double	8	1.7*10^{-308}.. 1.7*10^{308}	Virgulă mobilă dublă precizie
	long double	10	3.4*10^{-4932}..3.4*10^{4932}	Virgulă mobilă extra precizie

*lungimea exprimată în număr de octeți a reprezentării interne a tipurilor simple de date în C poate să difere (în special pentru tipul *int*)

b) Definirea de noi tipuri de date

- definirea de noi tipuri de date
- atribuirea unui alt nume unui tip predefinit sau definit anterior

```
typedef descriere_tip nume_utilizator;
```

Exemple:

```
typedef int INTREG;
typedef float REAL;
REAL x,y,z;
typedef char CARACTER;
```

c) Variabile

Variabilele:

- își modifică valoarea pe parcursul execuției programului;
- au asociate zone de memorie;
- domeniul de valabilitate este limitat la blocul în care s-a făcut declarația.

```
tip lista_variabile;
        sau
tip nume_variabila=valoare;
```

Exemple:

```
float a,b,c;
char k;
int x=50;
int y; y=50;
```

diferențe? – momentul la care se face inițializarea cu 50

Particularităţile unor tipuri simple de date

- **Discuții**

- **Exemple:**

```
unsigned char a,b;
……
a=100;
b=’Q’;
b=81;
a=a+b;
```

2.2. Constantele

a) Literalii întregi

b) Literalii reali

c) Literalii caracter

d) Literalii de tip șir de caractere

e) Constantele simbolice

f) Constantele obiect

a) Literalii întregi

Literalii întregi:

- reprezentaţi intern în virgulă fixă;

- pot fi exprimaţi în bazele de numerație:
  - 10 (forma implicită)
  - 8 (folosind prefixul *0* – zero)
  - 16 (folosind prefixul *0x* sau *0X*)

- se reprezintă:
  - conform tipului *int (*dacă este posibil), altfel
  - conform tipului *long*

- se poate folosi:
  - sufixul *l* sau *L* pentru reprezentare conform tipului *long*
  - sufixul *u* sau *U* pentru reprezentare conform tipului *unsigned*

Exemple: 10, 100, -8, …

b) Literalii reali

Literalii reali:

- sunt reprezentaţi intern în virgulă mobilă;

- se pot exprima sub formă:

  - matematică (**±întreg.fracţie**)

  - ştiinţifică (**±întreg.fracţieE±exponent**) – vezi **descriptorii de format** din tema 5. *Operații de I/E cu tastatura/monitorul*

- se reprezintă intern conform tipului *double;*

- se poate folosi:

  - sufixul *f* sau *F* pentru reprezentarea conform tipului *float*

  - sufixul *l* sau *L* pentru reprezentarea conform tipului *long double*

Exemple: 4.15, 18.9 …

c) Literalii caracter

Literalii caracter:

- se reprezintă intern prin codul ASCII al caracterului respectiv;

- se reprezintă pe un octet;

- pot participa în expresii cu valoarea lor numerică;

- pot fi:
  - caractere de control (coduri ASCII între 0 şi 31)
  - caractere direct imprimabile (coduri ASCII cuprinse între 32 şi 127)
  - caractere grafice (coduri ASCII cuprinse între 128 şi 255)

- **caractere de control** -> reprezentare folosind caracterul *backslash* (*secvenţă escape)*

Exemple

Literal	Cod ASCII	Denumire	Utilizare
'\b'	8	BS	Revenire cu un spaţiu (backspace)
'\t'	9	HT	Tab orizontal (9 spaţii)
'\n'	10	LF	Newline, corespunde perechii CR/LF – rând nou
'\v'	11	VT	Tab vertical
'\r'	13	CR	Poziţionare la începutul rândului (carriage return)

➢ **caractere direct imprimabile**

- reprezentare prin includerea caracterului între apostrofuri

- excepţie fac caracterele cu semnificație specială în C:
  - ‘ (*apostrof*),
  - " (*ghilimele*)
  - \ (*backslash*)

fiind precedate de caracterul \.

Exemple:

′B′, ′b′, ′7′

′\\′ (caracterul backslash)

′\′′ (caracterul apostrof)

′\"′ (caracterul ghilimele)

➢ **caractere grafice**

- se pot folosi secvenţele escape construite astfel: ′\ddd′, unde d este o cifră din sistemul de numeraţie octal (0÷7)

- această construcție poate fi folosită pentru a reprezenta orice caracter al setului ASCII

- **Exemple:**

′\a′ şi ′\7′ reprezintă caracterul BEL

′\"′ şi ′\42′ reprezintă caracterul ghilimele

′\377′ reprezintă caracterul cu codul ASCII 255

d) Literalii de tip șir de caractere

Literalul de tip șir de caractere:

- un șir de zero sau mai multe caractere, delimitate prin ghilimele (ghilimelele nu fac parte din șir)

- reprezentat intern prin codurile ASCII ale caracterelor, câte unul pe fiecare octet, la sfârșit adăugându-se caracterul nul (cod ASCII 0 – '\0')

- caracterul nu face parte dintr-un șir, el având rolul de terminator de șir

- un șir de caractere ocupă cu un octet mai mult decât numărul efectiv de caractere din componența sa

- **Exemple:**

"Acesta este un literal sir de caractere"

" " /*- şir de lungime 1 (caracterul spaţiu)*/

"" /*- şirul de lungime 0 (şirul vid)*/

e) Constantele simbolice

Constantele simbolice: NU se alocă memorie la compilare!!!

- sunt literali cărora li se asociază identificatori

```
#define nume_constanta valoare
```

- oferă avantaje:

  - nume sugestive, mai uşor de reţinut şi de utilizat

  - modificarea valorii constantei simbolice în definire pentru rularea succesivă a programului cu valori diferite ale unui literal utilizat

Exemple:

```
#define pi 3.141592653589
```

float a = pi+5; preprocesare -> a=3.141592653589 + 5 // nu se fac calcule

după execuţie a=8.1415…

```
#define e 2.718281828459
#define nume_facultate "CSIE"
#define raspuns 'D'
```

f) Constantele obiect

Constantele obiect:

- sunt variabile inițializate la declarare, pentru care **se rezervă memorie**, dar conținutul lor nu poate fi modificat pe parcursul execuției programului

```
const tip nume_constanta=valoare;
```

- **Exemplu:**

```
const int dim_vector=10;
```

2.3 Tipurile structurate

a) Tipul masiv

b) Tipul articol

c) Lucrul cu şiruri de caractere (tipul șir de caractere NU este implementat nativ în limbajul C)

a) Tipul masiv

Tipul masiv:

- desemnează o mulțime finită de elemente omogene constituită ca un tablou cu una, două sau mai multe dimensiuni

- structura în ansamblul ei nu trebuie să depășească zona de memorie maximă permisă pentru structurile de date

tip nume[dim1][dim2]…[dimn];

sau

tip nume[dim1][dim2]…[dimn]={{lista_const1}{lista_const2}…{lista_constn}};

- referirea elementelor se face prin acces direct folosind operatorii []

- **Exemple**: **int v[20]; float a[4][7]; char t[4][5][2]; //v[i], a[i][j], t[i][j][k]**

int x[10]={1,2,3,4,5}; // x = 1, 2, 3, 4, 5, 0, 0, 0, 0, 0 - implicit

b) Tipul articol

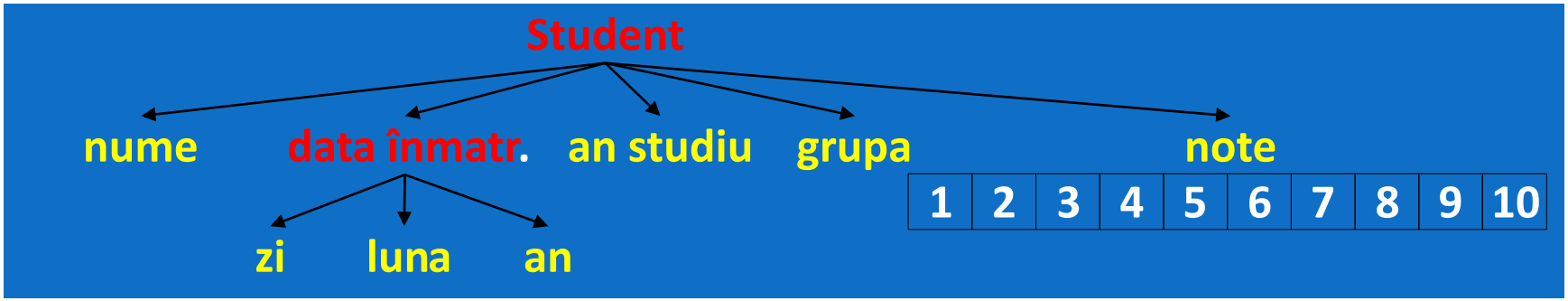
- Tip de dată
  - eterogen
  - cu acces direct la componente
  - există o relație de ordine ierarhică

- Reprezentare grafică
  - arbore
  - tabelară
  - blocuri

- Reprezentare internă
  - succesiuni de câmpuri elementare – *juxtapunere a câmpurilor elementare*

Reprezentare grafică

- Arbore: date de grup, date elementare (câmpuri)



- Tabelară

Student												
nume	data înmatr.			an studiu	grupa	note						
	zi	luna	an			1	2	3	4	5	6	7	8	9	10

Definire / declarare

```
struct NumeArticol {lista_campuri} var1, …, varn;
struct NumeArticol v, x[20];
```

```
typedef struct NumeArticol { lista_campuri } AliasNumeArticol;
AliasNumeArticol v, x[20];
```

Exemple

```
typedef struct { int zi, luna, an;
        } DATA;
typedef struct { char nume[40];
         DATA data_inmatr;
         int an_studiu;
         int grupa;
         int note[10];
        } STUDENT;

typedef struct { char nume[40];
         struct { int zi, luna, an;
              } data_inmatr;
         int an_studiu;
         int grupa;
         int note[10];
        } STUDENT;
```

Exemple. Variante

a) struct COMPLEX{float r, i;} a, b[100];

b) struct COMPLEX{float r, i;};
struct COMPLEX a, b[100];

c) struct COMPLEX{float r, i;};
COMPLEX a, b[100];

d) typedef struct {float r, i;} COMPLEX;
COMPLEX a, b[100]; /*struct COMPLEX a, b[100]; eroare*/

e) typedef struct COMPLEX {float r, i;};
struct COMPLEX a, b[100];

f) typedef struct COMPLEX{float r, i;};
COMPLEX a, b[100];

Memoria ocupată

Operatorul **sizeof()**

- sizeof(NumeArticol) = ?
  - dimensiunile câmpurilor membre
  - cerinţe de aliniere a datelor în memorie

- **Exemple**:

sizeof(COMPLEX)=? // 8

sizeof(STUDENT)=?

STUDENT v, x[20];

sizeof(v)=?

sizeof(x)=?

Memoria ocupată - exemple

```
struct art1
{char a1;int  b1;char a2;int
b2; char a3;int  b3; char a4;int
b4;};
```

sizeof(struct art1)=32

```
struct art2
{int  b1;int  b2;int  b3;int  b4;
char a1;char a2;char a3;char
a4; };
```

sizeof(struct art2)=20

sizeof(struct ST1)=16

```
struct ST1
{
   char ch1;
   short s;
   char ch2;
   long l;
   int i;
};
```

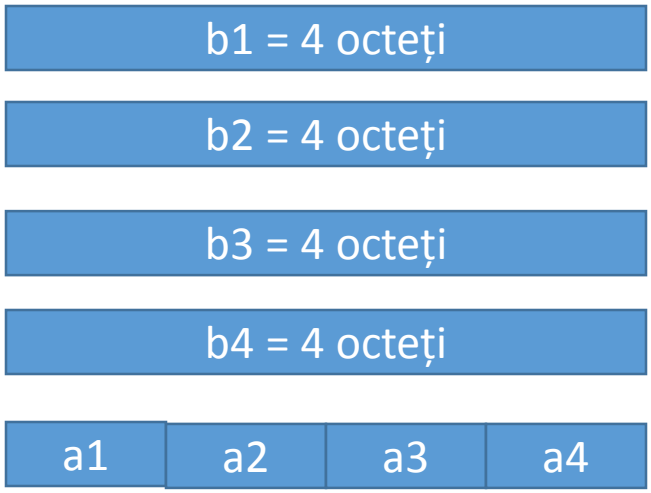
sizeof(struct ST2)=12

```
struct ST2
{
   long l;
   int i;
   short s;
   char ch1;
   char ch2;
} ;
```

Obs: alinierea structurii și alinierea câmpurilor depind de sistem/compilator

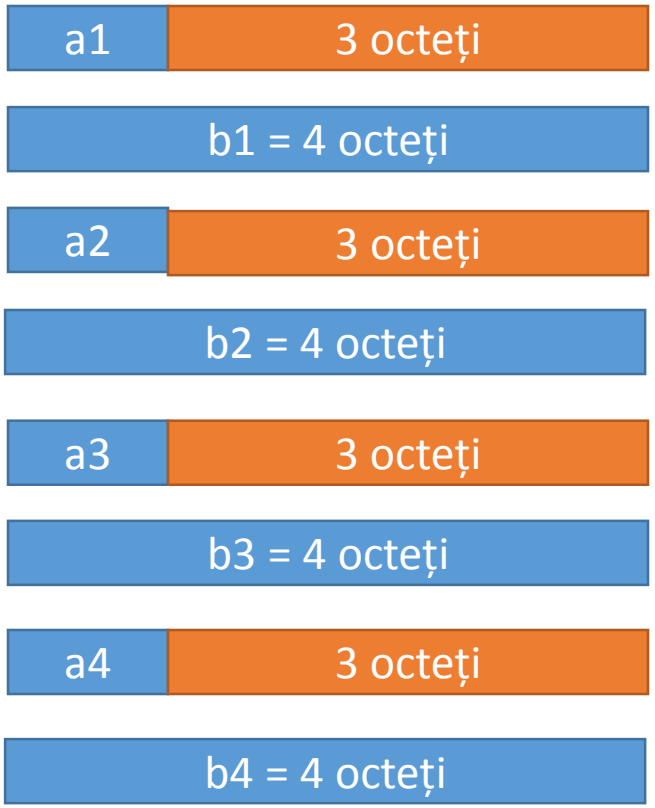
```
struct art2
{int  b1;int  b2;int  b3;int  b4;
char a1;char a2;char a3;char
a4; };
```

sizeof(struct art2)=20




```
struct art1
{char a1;int  b1;char a2;int
b2; char a3;int  b3; char a4;int
b4;};
```

sizeof(struct art1)=32



Obs: alinierea structurii și alinierea câmpurilor depind de sistem/compilator

Accesul la câmpuri

Operatorul de calificare

- punct **.**

  variabila.nume_camp

- săgeata **->**

  adresa_variabila->nume_camp

  (*adresa_variabila).nume_camp // vezi alocare dinamică

Operaţii

- Extragerea adresei (&)
- Atribuirea la nivel global
- Transmiterea ca argument în subprogram
- Încărcarea unei structuri
  - la declarare
  - prin atribuiri ulterioare
  - prin copiere în bloc

Articole cu structură variabilă

- ansamblu de variabile mutual exclusive

union nume_uniune {lista_campuri} var1, var2,…, varn;

- sizeof(nume_uniune)=?

Exemplu:

<table>
<tr><th colspan="11">Student</th></tr>
<tr><td rowspan="3">nume</td><td colspan="3" rowspan="2">data_inmatr.</td><td rowspan="3">an studiu</td><td rowspan="3">grupa</td><td colspan="5">forma de invatamant</td></tr>
<tr><td colspan="2">zi</td><td colspan="3">id</td></tr>
<tr><td>zi</td><td>luna</td><td>an</td><td>bursa</td><td>valoare</td><td rowspan="2">loc de munca</td><td colspan="3">data angajarii</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>zi</td><td>luna</td><td>an</td></tr>
</table>

Articole cu structură variabilă

```
struct STUDENT
      {char nume[40];
       struct {int zi, luna, an;} data_inmatr;  // Partea fixă
       int an_studiu;
       int grupa;
       char forma_inv; // câmpul selector: Z=frecventa, D=distanta
       union
          {struct {char bursa; // D=are bursa, N=nu are bursă (este cu taxa)
                   float valoare;} zi;
           struct {char loc_m[30];                    // Partea variabilă
                   struct {int zi, luna, an;} data_a;} id;
          } parte_var;
      };
struct STUDENT a={“Rusu Marius Ioan”, {1,10,2021}, 1, 1022,‘Z’, {‘D’,750}};
printf(“Valoare bursa =%6.2f”, a.parte_var.zi.valoare); // 750.00
```

Constante obiect de tip articol

- **Constantă obiect**

  const tip nume_const={lista_valori};

- **Exemplu:**

```
const STUDENT s={"Popescu",1,10,2021,1,1015,’Z’,'d',1000};
/*s.data_inmatr.zi=4; */
```

Transmiterea articolelor ca parametri

- **transferul prin valoare**

```
struct COMPLEX nume_fct(struct COMPLEX a)
{struct COMPLEX b;
 b.r = a.r * 2; b.i = a.i;
 return b;}

void main()
{struct COMPLEX x={1,2};
 struct COMPLEX y=nume_fct(x);}
```

Exemple

1. Să se construiască/declare în limbajul C articolul din reprezentarea tabelară de mai jos.

Cod magazin	Vânzări lunare			
	Luna 1	Luna 2	…	Luna 12
întreg	real	real	…	real

2. Să se construiască/declare în limbajul C articolul din reprezentarea tabelară de mai jos.

Cod produs	Număr materii prime	Materia primă 1		...	Materia primă 30	
		Cod	Norma de consum	...	Cod	Norma de consum
întreg	întreg	întreg	real	...	întreg	real

c) Lucrul cu șirurile de caractere

Şirurile de caractere:

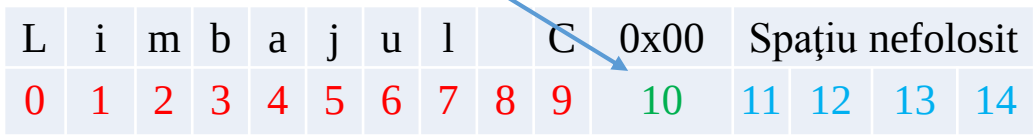
- se reprezintă ca masive unidimensionale (vectori) cu elemente de tipul *char*

- se reprezintă intern printr-o succesiune de octeți în care sunt memorate codurile ASCII ale caracterelor șirului

- ultimul octet conține caracterul NULL (cod ASCII 0 – sau ASCIIZ) nu face parte din șir

Exemple:

```
char c[15]= "Limbajul C";
char cuvant[]="este";
char x[4]="ABC";
char x[4]={'A','B','C',0x00};
```



Subprograme de bibliotecă pentru prelucrarea șirurilor de caractere (*string.h*)

- strcmp(sir1, sir2);
  - compară *sir1* cu *sir2* și returnează rezultat întreg:
    - negativ dacă sir1<sir2;
    - 0 dacă sir1=sir2;
    - pozitiv dacă sir1>sir2
- stricmp(sir1, sir2);
  - idem, ignorând diferențele dintre literele mici și literele mari (i – ignore)
- strncmp(sir1, sir2, n);
  - idem, dar comparația se face pentru cel mult primii n octeți
- strnicmp(sir1, sir2, n);
  - idem, ignorând diferențele dintre literele mici și literele mari, pentru cel mult primii n octeți

- strcat(dest, sursa);
  - concatenează șirul *sursa* la sfârșitul șirului *dest*
- strncat(dest, sursa, n);
  - concatenează primii n octeți din șirul *sursa* la sfârșitul șirului *dest*
- strcpy(dest, sursa);
  - copiază șirul *sursa* în șirul *dest*
- strncpy(dest, sursa, n);
  - copiază primii n octeți din șirul *sursa* în șirul *dest*
- strlen(sir);
  - returnează lungimea șirului *sir*

Biblioteci standard: *string.h, stdlib.h, math.h, stdio.h, conio.h .....*