



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,
2016-2017

Cursul 5

Balul bobocilor

- balul organizat de AS-MI are afișul următor



Recapitulare – cursul trecut

1. Operatori și expresii. Conversii
2. Instrucțiuni de control
3. Directive de preprocesare. Macrodefiniții
4. Etapele realizării unui program în C

Programa cursului

□ Introducere

- Algoritmi.
- Limbaje de programare.
- Introducere în limbajul C. Structura unui program C.
- Complexitatea algoritmilor.

□ Fundamentele limbajului C

- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Instrucțiuni de control
- Directive de preprocesare. Macrodefiniții.
- Funcții de citire/scriere.
- Etapele realizării unui program C.

□ Tipuri derivate de date

- Tablouri. Siruri de caractere.
- Structuri, uniuni, câmpuri de biți, enumerări.
- Pointeri.

□ Funcții (1)

- Declarație și definire. Apel. Metode de transmisie a parametrilor.
- Pointeri la funcții.

□ Tablouri și pointeri

- Legătura dintre tablouri și pointeri
- Aritmetică a pointerilor
- Alocarea dinamică a memoriei
- Clase de memorare

□ Siruri de caractere

- Funcții specifice de manipulare.

□ Fișiere text și fișiere binare

- Funcții specifice de manipulare.

□ Structuri de date complexe și autoreferite

- Definire și utilizare

□ Funcții (2)

- Funcții cu număr variabil de argumente.
- Preluarea argumentelor funcției main din linia de comandă.
- Programare generică.

□ Recursivitate

Cursul de azi

1. Pointeri
2. Funcții de citire/scriere
3. Tablouri. Siruri de caractere.
4. Structuri, uniuni, câmpuri de biți, enumerări.

Pointeri

- **pointer** = tip de date derivat folosit pentru manipularea adreselor de memorie
- **sintaxa**

tip *nume_variabilă;

 - **tip** = tipul de bază al variabilei de tip pointer nume_variabilă
 - ***** = operator de indirectare
 - **nume_variabila** = variabila de tip pointer care poate lua ca valori adrese de memorie
- **cel mai puternic mecanism de accesare a memoriei în C**

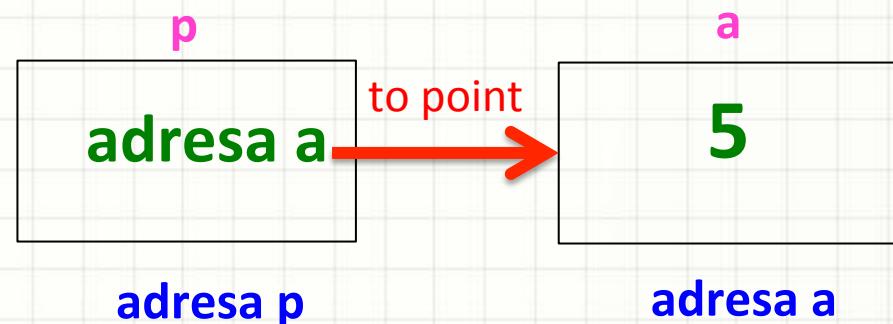
Pointeri

- pointer = tip de date derivat folosit pentru manipularea adreselor de memorie
- operatori pentru manipularea adreselor de memorie:
 - & - operatorul de referențiere
 - **&variabila** – furnizează adresa variabilei respective
 - * - operatorul de dereferențiere
 - ***variabila_de_tip_pointer** – furnizează valoarea aflată la adresa de memorie stocată în variabila_de_tip_pointer

Pointeri

□ exemplu:

```
int a=5  
int *p;  
p = &a;
```



main.c

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    int a=5,*p;
    p = &a;

    printf("Adresa lui a este: %d \n",&a);
    printf("Valoarea stocata la adresa lui a este: %d \n",a);

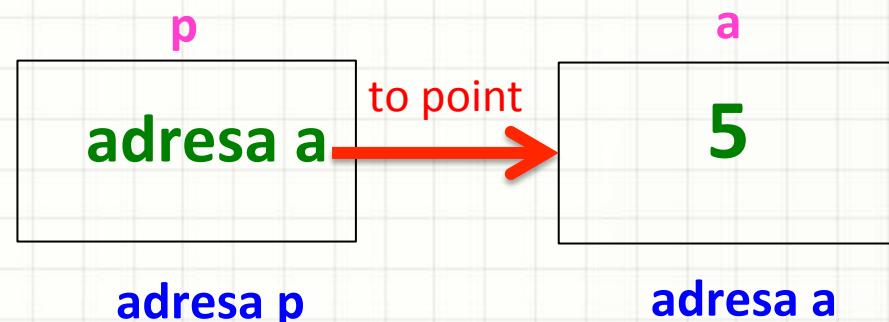
    printf("Adresa lui p este: %d \n",&p);
    printf("Valoarea stocata la adresa lui p este: %d \n",p);
    printf("Valoarea variabilei a carei adresa e stocata in p este %d \n",*p);

    return 0;
}
```

Pointeri

□ exemplu:

```
int a=5  
int *p;  
p = &a;
```



main.c

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    int a=5,*p;
    p = &a;

    printf("Adresa lui a este: %d \n",&a);
    printf("Valoarea stocata la adresa lui a este: %d \n",a);

    printf("Adresa lui p este: %d \n",&p);
    printf("Valoarea stocata la adresa lui p este: %d \n",p);
    printf("Valoarea variabilei a carei adresa e stocata in p este %d \n",*p);

    return 0;
}
```

Adresa lui a este: 1606416748
Valoarea stocata la adresa lui a este: 5
Adresa lui p este: 1606416736
Valoarea stocata la adresa lui p este: 1606416748
Valoarea variabilei a carei adresa e stocata in p este 5
Process returned 0 (0x0) execution time : 0.012 s
Press ENTER to continue.

Pointeri

- exemplu: modificarea valorii unei variabile prin dereferențiere

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6
7     int a=5,*p;
8     p = &a;
9
10    printf("Valoarea stocata la adresa lui a este: %d \n",a);
11
12    a += 10; //acces direct
13    printf("Valoarea stocata la adresa lui a este: %d \n",a);
14
15    *p += 20; //acces indirect
16    printf("Valoarea stocata la adresa lui a este: %d \n",a);
17
18    return 0;
19
20 }
```

```
Valoarea stocata la adresa lui a este: 5
Valoarea stocata la adresa lui a este: 15
Valoarea stocata la adresa lui a este: 35
```

```
Process returned 0 (0x0) execution time : 0.006 s
Press ENTER to continue.
```

Pointeri

- exemplu: pointeri neinitializați p

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6
7     int a=5,*p;
8     printf("a= %d\n",a);
9     *p = 10;
10    printf("a= %d \n",a);
11    return 0;
12}
13
```



a= 5

Process returned -1 (0xFFFFFFFF) execution time : 0.027 s
Press ENTER to continue.

Pointeri

- exemplu: spațiul de memorie ocupat de o variabilă de tip pointer

```
main.c ✘
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6
7     int a=5,*p;
8     p = &a;
9     printf("Variabila a ocupa %d octeti \n",sizeof(a));
10    printf("Variabila de tip pointer p ocupa %d octeti \n",sizeof(p));
11
12    return 0;
13
14 }
```

```
Variabila a ocupa 4 octeti
Variabila de tip pointer p ocupa 8 octeti

Process returned 0 (0x0)    execution time : 0.005 s
Press ENTER to continue.
```

Pointeri

□ adrese de memorie

- În C există un specificator de format special (%p) pentru tipărirea valorilor reprezentând adresele de memorie.

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6
7     int x;
8
9     printf("Adresa lui x este %p \n",&x);
10    printf("Adresa lui x este %x \n",&x);
11    printf("Adresa lui x este %d \n",&x);
12
13    return 0;
14 }
15
```

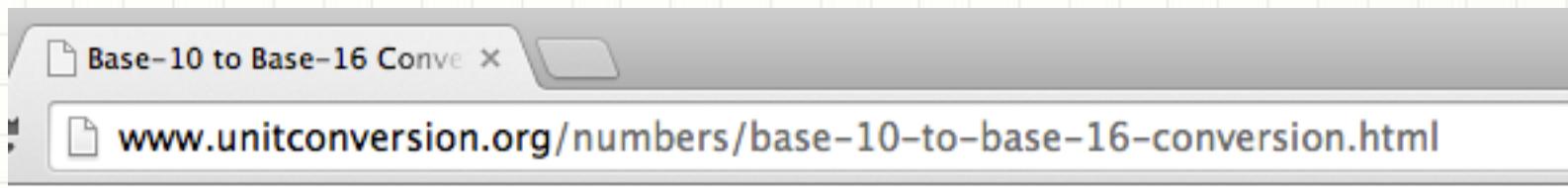
```
Adresa lui x este 0x7fff5fbff96c
Adresa lui x este 5fbff96c
Adresa lui x este 1606416748

Process returned 0 (0x0)  execution time : 0.008 s
Press ENTER to continue.
```

Pointeri

□ adrese de memorie

- În C există un specificator de format special (%p) pentru tipărirea valorilor reprezentând adresele de memorie.



base-10:

1606416748

base-16:

5FBFF96C

```
Adresa lui x este 0x7fff5fbff96c
Adresa lui x este 5fbff96c
Adresa lui x este 1606416748
```

```
Process returned 0 (0x0)    execution time : 0.008 s
Press ENTER to continue.
```

Cursul de azi

1. Pointeri
2. Funcții de citire/scriere
3. Tablouri. Siruri de caractere.
4. Structuri, uniuni, câmpuri de biți, enumerări.

Functii de citire și scriere

- operații de **citire** și **scriere** în C:
 - de la **tastatură** (stdin) și la **ecran** (stdout);
 - **prin fișiere**;
 - efectuate cu ajutorul funcțiilor de bibliotecă
- funcții pentru **citirea de la tastatură** și **scrierea la ecran**
 - fără formatare: **getchar**, **putchar**, **getch**, **getche**, **putch**, **gets**, **puts**
 - cu formatare: **scanf**, **printf**
 - incluse în bibliotecile **stdio.h** (**getchar**, **putchar**, **gets**, **puts**, **scanf**, **printf**) sau **conio.h** (**getch**, **getche**, **putch**)
 - CODE::BLOCKS nu include biblioteca **conio.h**

Functiile getchar și putchar

- operatii de **citire** și **scriere** a caracterelor:
 - **int getchar(void)** - citește un caracter de la tastatură. Așteaptă până este apasată o tastă și returnează valoarea sa → tasta apăsată are imediat ecou pe ecran.
 - **int putchar(int c)** - scrie un caracter pe ecran în poziția curentă a cursorului
 - fișierul antet pentru aceste funcții este **stdio.h.**

Functiile getchar și putchar

□ exemplu:

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6     int optiune;
7
8     printf("Alegeti DA sau NU. Optiunea dumneavoastră este : ");
9     optiune = getchar();
10    putchar(optiune);
11
12    return 0;
13
14 }
15
```

```
Alegeti DA sau NU. Optiunea dumneavoastră este : DA
D
Process returned 0 (0x0)    execution time : 1.321 s
Press ENTER to continue.
```

Functiile gets și puts

- operații de **citire și scriere** a sirurilor de caractere:
 - **char *gets(char *s)** – citește caractere din **stdin** și le depune în zona de date de la adresa s, până la apăsarea tastei Enter. În sir, tastei Enter îi va corespunde caracterul '\0'.
 - dacă operația de citire reușește, funcția întoarce adresa sirului, altfel valoarea **NULL** (= 0).
 - **int puts(const char *s)** – scrie pe ecran sirul de la adresa s sau o constantă sir de caractere și apoi trece la linie nouă.
 - dacă operația de scriere reușește, funcția întoarce ultimul caracter, altfel valoarea **EOF** (-1).
 - fișierul antet pentru aceste funcții este **stdio.h**

Functiile gets și puts

❑ exemplu:

```
main.c ✘
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6     char sir[10];
7
8     printf("Ce zi e astazi: ");
9     gets(sir);
10    puts(sir);
11
12    puts("Alegeți DA sau NU. Optiunea dumneavoastră este: ");
13    gets(sir);
14    puts(sir);
15
16
17    return 0;
18
19
20 }
```

```
Ce zi e astazi: joi
joi
Alegeți DA sau NU. Optiunea dumneavoastră este:
DA
DA

Process returned 0 (0x0)  execution time : 3.418 s
Press ENTER to continue.
```

De ce să nu folosiți funcția gets

- **char *gets(char *s)**
- primește ca input numai un buffer (**s**), nu stim dimensiunea lui
- problema de buffer overflow: citim în **s** mai mult decât dimensiunea lui, **gets** nu ne impiedică, scrie datele în alta parte
- folositi fgets: **char *fgets(char *s, int size, FILE *stream)**
 - **fgets(buffer, sizeof(buffer), stdin);**
- în standardul C11 funcția gets este eliminată

Functiile printf și scanf

- funcții de citire (**scanf**) și scriere (**printf**) cu formatare;
- formatarea specifică conversia datelor de la reprezentarea externă în reprezentarea internă (**scanf**) și invers (**printf**);
- formatarea se realizează pe baza descriptorilor de format
 - **%[flags][width][.precision][length]specifier**
 - detalii aici: <http://www.cplusplus.com/reference/cstdio/printf/>

Functiile printf și scanf

- formatarea se realizează pe baza descriptorilor de format
 - %[flags][width][.precision][length]specifier
 - detalii aici: <http://www.cplusplus.com/reference/cstdio/printf/>
 - specifier

Specificator de format	Reprezentare
%c	caracter
%s	șir de caractere
%d, %i	întreg în zecimal
%u	întreg în zecimal fără semn
%o	întreg în octal
%x	întreg în hexazecimal fără semn (litere mici)
%X	întreg în hexazecimal fără semn (litere mari)
%f	număr real în virgulă mobilă
%e, %E	notație științifică – o cifră la parte întreagă
%ld, %li, %lu, %lo, %lx	cu semnificațiile de mai sus, pentru întregi lungi
%p	pointer

Functia printf

□ prototipul funcției:

*int printf(const char *format, argument1, argument2, ...);*

unde:

- **format** este un sir de caractere ce definește textele și formatele datelor care se scriu pe ecran
- **argument1, argument2,...** sunt expresii. Valorile lor se scriu pe ecran conform specificatorilor de format prezenți în format
- functia **printf** întoarce numărul de octeți transferați sau EOF (-1) în caz de eșec.

Functia printf

□ exemplu:

main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5
6     char sir[10] = "Azi e joi";
7     printf("Primul caracter din sirul \"%s\" este %c \n",sir,sir[0]);
8     int x = 1234;
9     printf("Reprezentare lui x in baza 10: x=%d\n",x);
10    printf("Reprezentare lui x in baza 8: x=%o\n",x);
11    printf("Reprezentare lui x in baza 16 (litere mici): x=%x\n",x);
12    printf("Reprezentare lui x in baza 16 (litere mari): x=%X\n",x);
13
14    float y = 12.34;
15    printf("Reprezentare lui y ca numar real: y=%f\n",y);
16    printf("Reprezentare lui y in notatie stiintifica: y=%e\n",y);
17    printf("Reprezentare lui y in notatie stiintifica: y=%E\n",y);
18
19    return 0;
20 }
21 
```

Primul caracter din sirul "Azi e joi" este A
Reprezentare lui x in baza 10: x=1234
Reprezentare lui x in baza 8: x=2322
Reprezentare lui x in baza 16 (litere mici): x=4d2
Reprezentare lui x in baza 16 (litere mari): x=4D2
Reprezentare lui y ca numar real: y=12.340000
Reprezentare lui y in notatie stiintifica: y=1.234000e+01
Reprezentare lui y in notatie stiintifica: y=1.234000E+01

Functia printf

□ exemplu:

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6
7     int a = printf("Ce zi e astazi: ");
8     printf("\n a = %d \n",a);
9
10    a = printf("Alegeti DA sau NU. Optiunea dumneavoastră este: ");
11    printf("\n a = %d \n",a);
12
13    return 0;
14 }
15
```

Ce zi e astazi:
a = 16
Alegeti DA sau NU. Optiunea dumneavoastră este:
a = 48

Process returned 0 (0x0) execution time : 0.005 s
Press ENTER to continue.

Modelatori de format

- mulți specicatori de format pot accepta modelatori care modifică ușor semnificația lor:
 - alinierea la stânga
 - minim de mărime a câmpului
 - numărul de cifre zecimale

- modelatorul de format se află între semnul procent și codul pentru format:
 - caracterul ‘–’ specifică aliniere la stânga;
 - sir de cifre zecimale specifică dimensiunea câmpului pentru afişare
 - caracterul ‘.’ urmat de cifre specifică precizia reprezentării

Modelatorul flags

- formatarea se realizează pe baza descriptorilor de format
 - `%[flags][width][.precision][length]specifier`
 - detalii aici: <http://www.cplusplus.com/reference/cstdio/printf/>
 - flags

flags	description
-	Left-justify within the given field width; Right justification is the default (see <i>width</i> sub-specifier).
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is going to be written, a blank space is inserted before the value.
#	Used with o, x or X specifiers the value is preceded with 0, 0x or 0X respectively for values different than zero. Used with a, A, e, E, f, F, g or G it forces the written output to contain a decimal point even if no more digits follow. By default, if no digits follow, no decimal point is written.
0	Left-pads the number with zeroes (0) instead of spaces when padding is specified (see <i>width</i> sub-specifier).

Modelatorul width

- formatarea se realizează pe baza descriptorilor de format
 - `%[flags][width][.precision][length]specifier`
 - detalii aici: <http://www.cplusplus.com/reference/cstdio/printf/>
 - **width**

width	description
(number)	Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The <i>width</i> is not specified in the <i>format</i> string, but as an additional integer value argument preceding the argument that has to be formatted.

Modelatorul precision

- formatarea se realizează pe baza descriptorilor de format
 - `%[flags][width][.precision][length]specifier`
 - detalii aici: <http://www.cplusplus.com/reference/cstdio/printf/>
 - precision

.precision	description
.number	For integer specifiers (d, i, o, u, x, x): <i>precision</i> specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A <i>precision</i> of 0 means that no character is written for the value 0. For a, A, e, E, f and F specifiers: this is the number of digits to be printed after the decimal point (by default, this is 6). For g and G specifiers: This is the maximum number of significant digits to be printed. For s: this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered. If the period is specified without an explicit value for <i>precision</i> , 0 is assumed.
.*	The <i>precision</i> is not specified in the <i>format</i> string, but as an additional integer value argument preceding the argument that has to be formatted.

Modelatorul length

- formatarea se realizează pe baza descriptorilor de format
 - %[flags][width][.precision][length]specifier
 - detalii aici: <http://www.cplusplus.com/reference/cstdio/printf/>
 - length

length	specifiers							
	d i	u o x X	f F e E g G a A	c	s	p	n	
(none)	int	unsigned int	double	int	char*	void*	int*	
hh	signed char	unsigned char						signed char*
h	short int	unsigned short int						short int*
l	long int	unsigned long int		wint_t	wchar_t*			long int*
ll	long long int	unsigned long long int						long long int*
j	intmax_t	uintmax_t						intmax_t*
z	size_t	size_t						size_t*
t	ptrdiff_t	ptrdiff_t						ptrdiff_t*
L			long double					

Modelatori de format pentru printf

exemplu:

```
main.c
001 #include <stdio.h>
002 #include <stdlib.h>
003
004
005 int main(){
006     double numar;
007     numar = 10.1234;
008
009     printf("numar=%f\n",numar);
010     printf("numar=%10f\n",numar);
011     printf("numar=%012f\n",numar);
012
013     printf("%.4f\n",123.1234567);
014     printf("%3.8d\n",1000);
015     printf("%10d\n",1000);
016     printf("%-10d\n",1000);
017     printf("%10.15s\n","Acesta este un test simplu");
018
019     return 0;
020 }
021 }
```

```
numar=10.123400
numar= 10.123400
numar=00010.123400
123.1235
00001000
          1000
1000
Acesta este un
```

Functia printf

□ exemplu:

```
printf("valoarea lui x este: %-4.2f\n",3.14);
printf("x=%i, y=%f, x=%o, x=%#x\n",15,3.14,15,15);
printf("c= %c, c=%d\n", '%', '%');
printf("sir de caractere: %s\n", "ana are mere");
printf("\\ \" \' \\n");
```

```
| valoarea lui x este: 3.14
| x=15, y=3.140000, x=17, x=0xf
| c= %, c=37
| sir de caractere: ana are mere
| \ " '
```

Functia scanf

- **prototipul functiei:**

int scanf(const char * format ,adresa1, adresa2, ...);

unde:

- **format** este un sir de caractere ce defineste textele si formatele datelor care se citesc de la tastatura
- **adresa1, adresa2,...** sunt adresele zonelor din memorie in care se pастreaza datele citite după ce au fost convertite din reprezentarea lor externă în reprezentare internă.
- functia **scanf** întoarce numărul de câmpuri citite și depuse la adresele din listă. Dacă nu s-a stocat nici o valoare, funcția întoarce 0.

Functia scanf

- sirul de formatare (format) poate include următoarele elemente:
 - spațiu alb: funcția citește și ignoră spațiile albe (spațiu, tab, linie nouă) înaintea următorului caracter diferit de spațiu
 - un singur spațiu în sirul de formatare se suprapune asupra oricărora spații din sirul introdus, inclusiv asupra nici unui spațiu
 - caracter diferit de spațiu, cu excepția caracterului %: funcția citește următorul caracter de la intrare și îl compară cu caracterul specificat în sirul de formatare
 - dacă se potrivește, funcția are succes și trece mai departe la citirea următorului caracter din intrare
 - dacă nu se potrivește, funcția eșuează și lasă următoarele caractere din intrare nepreluate

Functia scanf

- exemplu:

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6
7     int n,a;
8     float m;
9
10    scanf("n=%d m=%f",&n,&m);
11    printf("n=%d\nm=%f\n",n,m);
12
13    printf("n=");
14    scanf("%d",&n);
15
16    printf("m=");
17    a = scanf("%f",&m);
18    printf("a = %d \n",a);
19
20    return 0;
21 }
```

Trebuie să îi dau inputul aşa

n=25 m=3.2
n=25
m=3.200000
n=100
m=i37
a = 0

Process returned 0 (0x0) execution time : 14.176
Press ENTER to continue.

Functia scanf

exemplu:

```
main.c ✘
01 #include <stdio.h>
02 #include <stdlib.h>
03
04
05 int main(){
06
07     char a,b,c,sir[80];
08
09     scanf("%c%c%c",&a,&b,&c);
10    printf("a=%c\nb=%c\nc=%c\n",a,b,c);
11    printf("Introduceti un sir:");
12    scanf("%s",sir); ←
13    printf("Sirul tastat este %s",sir);
14
15    return 0;
16
17 }
```

'sir' e o adresa de memorie (pointer constant).
sir = &sir = &sir[0]

```
azi
a=a
b=z
c=i
Introduceti un sir:maine
Sirul tastat este maine
Process returned 0 (0x0)  execution time : 3.482 s
Press ENTER to continue.
```

Functia scanf

□ exemplu:

```
int main()
{
    int a, b;
    char c;
    for(;;)
    {
        printf("Introduceti 2 numere intregi\n");
        if (scanf("%d%d", &a, &b) == 2)
            break;
        else
            while (c = getchar() != '\n' && c != EOF);
    }
    printf("am citit 2 numere: %d si %d\n", a, b);
    return 0;
}
```

```
Introduceti 2 numere intregi
3 a
Introduceti 2 numere intregi
4 2 3
am citit 2 numere: 4 si 2
```

Cursul de azi

1. Pointeri
2. Funcții de citire/scriere
3. Tablouri. Siruri de caractere.
4. Structuri, uniuni, câmpuri de biți, enumerări.

Tablouri unidimensionale

- **sintaxă:**

tip nume_tablou [dimensiune];

- **exemple:**

double v[100];

0.3	-1.2	10	5.7	...	0.2	-1.5	1
0	1	2	3		97	98	99

v[3] = 5.7;

int a[5];

3	-12	10	7	1
0	1	2	3	4

a[0] = 3;

char c[34];

A	&	*	+	...	c	M	#
0	1	2	3		31	32	33

c[1] = '&';

- În C, primul element al unui tablou are indicele 0.

Tablouri unidimensionale

- definiție: set de valori de același tip memorat la adrese succesive de memorie.

```
tablou1.c
1 #include <stdio.h>
2
3 int main(){
4     int a[5],i;
5     for(i=0;i<5;i++)
6         printf("Adresa elementului %d din tabloul a este %p \n",i,&a[i]);
7         printf("*****\n");
8
9     double v[100];
10    for(i=0;i<3;i++)
11        printf("Adresa elementului %d din tabloul v este %p \n",i,&v[i]);
12        printf("*****\n");
13
14    char c[34];
15    for(i=0;i<4;i++)
16        printf("Adresa elementului %d din tabloul c este %p \n",i,&c[i]);
17        printf("*****\n");
18
19    return 0;
20
21 }
```

Tablouri unidimensionale

- definiție: set de valori de același tip memorat la adrese succesive de memorie.

```
tablou1.c   
1 #include <stdio.h>  
2  
3 int main(){  
4     int a[5],i;  
5     for(i=0;i<5;i++)  
6         printf("Adresa elementului %d din tabloul a este %p \n",i,&a[i]);  
7     printf("*****\n");  
8  
9     double v[100];  
10    for(i=0;i<3;i++)  
11        printf("Adresa elementului %d din tabloul v este %p \n",i,&v[i]);  
12    printf("*****\n");  
13  
14    char c[34];  
15    for(i=0;i<4;i++)  
16        printf("Adresa elementului %d din tabloul c este %p \n",i,&c[i]);  
17    printf("*****\n");  
18  
19    return 0;  
20  
21
```

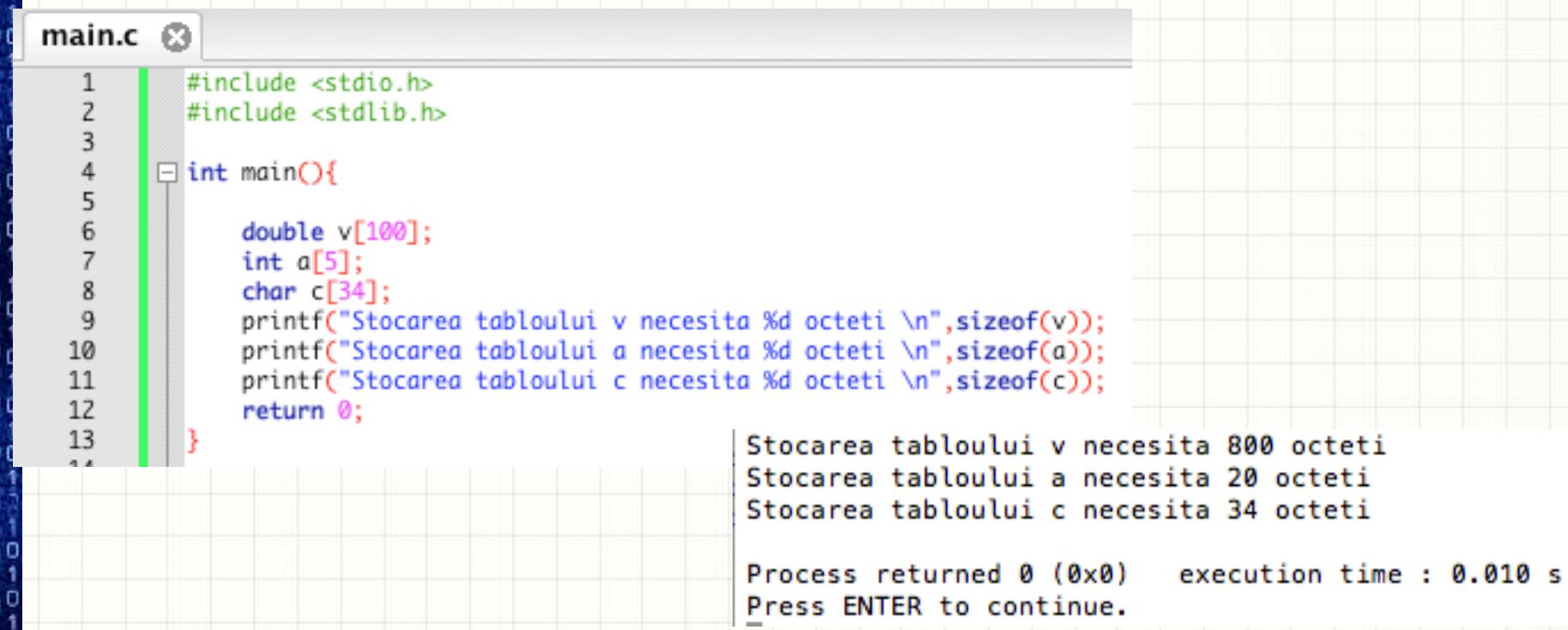
Tablouri unidimensionale

- definiție: set de valori de același tip memorat la adrese succesive de memorie.

- memorie:
 - cantitatea de memorie necesară pentru stocarea unui tablou este direct proporțională cu tipul de date și mărimea sa.
 - tip nume [dimensiune] → **sizeof(nume) = sizeof (tip) * dimensiune ;**

Tablouri unidimensionale

- **memorie:**
 - cantitatea de memorie necesară pentru stocarea unui tablou este direct proporțională cu tipul de date și mărimea sa.
 - **tip nume [dimensiune] → `sizeof(nume) = sizeof (tip) * dimensiune ;`**



```
main.c x
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5
6     double v[100];
7     int a[5];
8     char c[34];
9     printf("Stocarea tabloului v necesita %d octeti \n",sizeof(v));
10    printf("Stocarea tabloului a necesita %d octeti \n",sizeof(a));
11    printf("Stocarea tabloului c necesita %d octeti \n",sizeof(c));
12
13    return 0;
14 }
```

```
Stocarea tabloului v necesita 800 octeti
Stocarea tabloului a necesita 20 octeti
Stocarea tabloului c necesita 34 octeti

Process returned 0 (0x0)  execution time : 0.010 s
Press ENTER to continue.
```

Tablouri unidimensionale

□ exemplu de initializare

```
1 #include <stdio.h>
2
3 int main(){
4
5     int a[3], x=100;
6     printf("x=%d\n",x);
7     a[0] = a[1] = a[2] = a[3] = 17;
8
9     printf("x=%d\n",x);
10
11    return 0;
12 }
```

x=100
x=17

Tablouri unidimensionale

□ exemplu de initializare

```
1 #include <stdio.h>
2
3 int main(){
4
5     int a[3], x=100;
6     printf("x=%d\n",x);
7     a[0] = a[1] = a[2] = a[3] = 17;
8
9     printf("x=%d\n",x);
10
11    printf("Adresa lui a[2] este %p \n",&a[2]);
12    printf("Adresa lui x este %p \n",&x);
13
14    return 0;
15 }
```

x=100
x=17
Adresa lui a[2] este 0x7fff5fbff988
Adresa lui x este 0x7fff5fbff98c

Tablouri unidimensionale

□ exemplu de initializare

```
main.c ✘
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6
7     int nrElemente,i;
8     int a1[3] = {1,2,3};
9     printf("Tabloul a1 are %d elemente\n",nrElemente = sizeof(a1)/sizeof(int));
10    for(i=0;i<nrElemente;i++)
11        printf("a1[%d]=%d \t",i,a1[i]);           Tabloul a1 are 3 elemente
12
13
14     int a2[6] = {1,2,3};
15     printf("Tabloul a2 are %d elemente\n",nrElemente = sizeof(a2)/sizeof(int));
16     for(i=0;i<nrElemente;i++)
17         printf("a2[%d]=%d \t",i,a2[i]);           a1[0]=1          a1[1]=2          a1[2]=3
18
19
20     int a3[6] = {0};
21     printf("Tabloul a3 are %d elemente\n",nrElemente = sizeof(a3)/sizeof(int));
22     for(i=0;i<nrElemente;i++)
23         printf("a3[%d]=%d \t",i,a3[i]);           a2[0]=1          a2[1]=2          a2[2]=3          a2[3]=0          a2[4]=0          a2[5]=0
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
```

Tablouri unidimensionale

□ exemplu de initializare

```
29     int a4[6] = {10};  
30     printf("Tabloul a4 are %d elemente\n",nrElemente = sizeof(a4)/sizeof(int));  
31     for(i=0;i<nrElemente;i++)  
32         printf("a4[%d]=%d \t",i,a4[i]);
```

```
Tabloul a4 are 6 elemente  
a4[0]=10      a4[1]=0      a4[2]=0      a4[3]=0      a4[4]=0      a4[5]=0
```

```
36     int a5[6] = {1,[2]=4,[3]=7,9};  
37     printf("Tabloul a5 are %d elemente\n",nrElemente = sizeof(a5)/sizeof(int));  
38     for(i=0;i<nrElemente;i++)  
39         printf("a5[%d]=%d \t",i,a5[i]);
```

```
Tabloul a5 are 6 elemente  
a5[0]=1      a5[1]=0      a5[2]=4      a5[3]=7      a5[4]=9      a5[5]=0
```

```
43     int a6[6] = {1,[2]=4,[1]=7,9};  
44     printf("Tabloul a6 are %d elemente\n",nrElemente = sizeof(a6)/sizeof(int));  
45     for(i=0;i<nrElemente;i++)  
46         printf("a6[%d]=%d \t",i,a6[i]);
```

```
Tabloul a6 are 6 elemente  
a6[0]=1      a6[1]=7      a6[2]=9      a6[3]=0      a6[4]=0      a6[5]=0  
Process returned 6 (0x6)  execution time : 0.004 s
```

Tablouri bidimensionale

- **sintaxa**

tip nume_variabila [dimensiune1][dimensiune2];

- **exemplu**

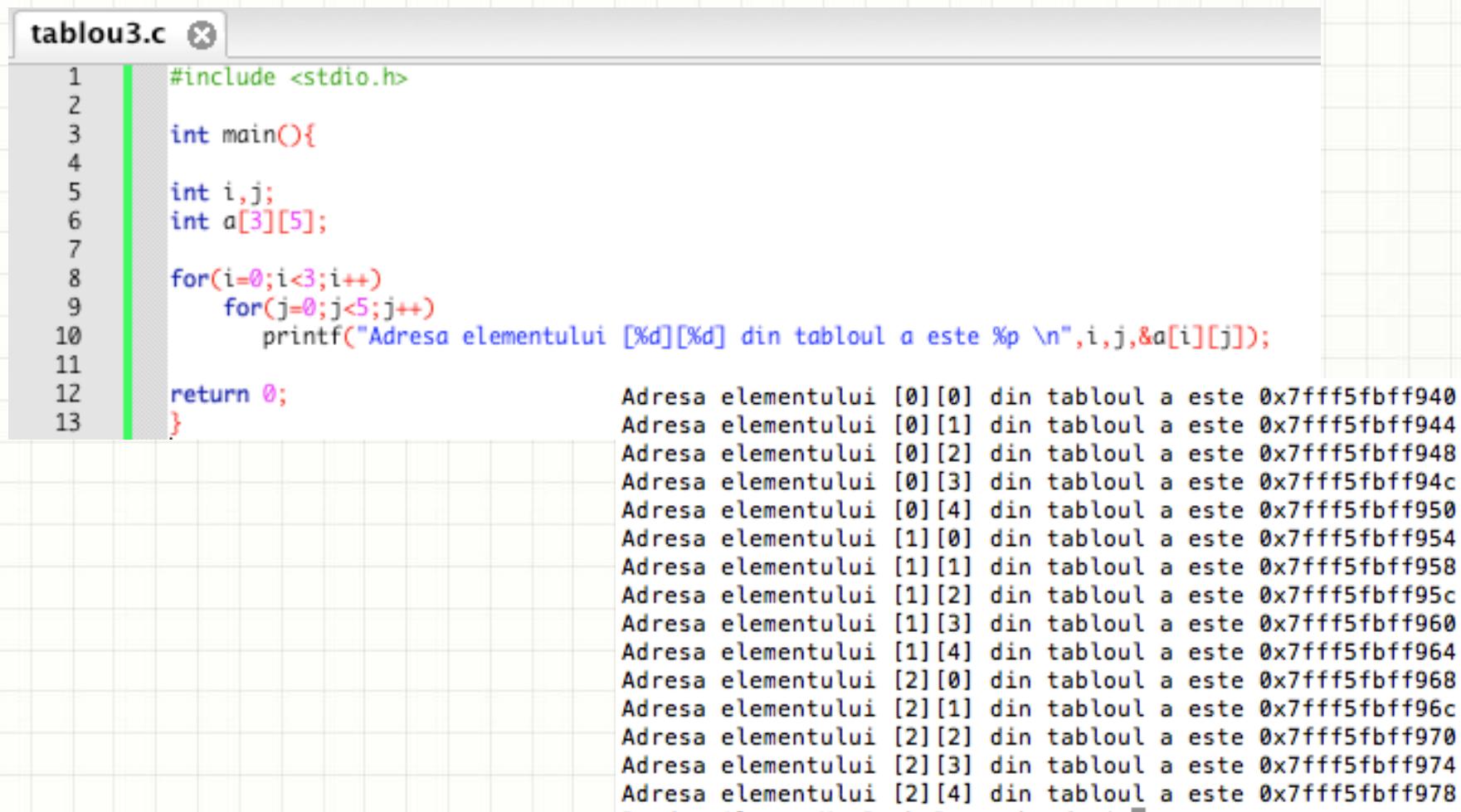
```
int a[3][5];
```

```
a[1][4] = 41;
```

0	3	-12	10	7	1
1	10	2	0	-7	41
2	-3	-2	0	0	2
	0	1	2	3	4

Tablouri bidimensionale

- definiție: set de valori de același tip memorat la adrese succesive de memorie.



The screenshot shows a code editor window titled "tablou3.c". The code is a simple C program that declares a 3x5 integer matrix 'a'. It uses nested loops to iterate through each element and prints its address using the printf function. The output shows the memory addresses for each element in row-major order.

```
#include <stdio.h>
int main(){
    int i,j;
    int a[3][5];
    for(i=0;i<3;i++)
        for(j=0;j<5;j++)
            printf("Adresa elementului [%d][%d] din tabloul a este %p \n",i,j,&a[i][j]);
    return 0;
}
```

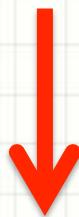
Adresa elementului [0][0] din tabloul a este 0xffff5fbff940
Adresa elementului [0][1] din tabloul a este 0xffff5fbff944
Adresa elementului [0][2] din tabloul a este 0xffff5fbff948
Adresa elementului [0][3] din tabloul a este 0xffff5fbff94c
Adresa elementului [0][4] din tabloul a este 0xffff5fbff950
Adresa elementului [1][0] din tabloul a este 0xffff5fbff954
Adresa elementului [1][1] din tabloul a este 0xffff5fbff958
Adresa elementului [1][2] din tabloul a este 0xffff5fbff95c
Adresa elementului [1][3] din tabloul a este 0xffff5fbff960
Adresa elementului [1][4] din tabloul a este 0xffff5fbff964
Adresa elementului [2][0] din tabloul a este 0xffff5fbff968
Adresa elementului [2][1] din tabloul a este 0xffff5fbff96c
Adresa elementului [2][2] din tabloul a este 0xffff5fbff970
Adresa elementului [2][3] din tabloul a este 0xffff5fbff974
Adresa elementului [2][4] din tabloul a este 0xffff5fbff978

Tablouri bidimensionale

- definiție: set de valori de același tip memorat la adrese succesive de memorie.

3	-12	10	7	1
10	2	0	-7	41
-3	-2	0	0	2

0 1 2 3 4



Adresa elementului [0][0] din tabloul a este 0xffff5fbff940
Adresa elementului [0][1] din tabloul a este 0xffff5fbff944
Adresa elementului [0][2] din tabloul a este 0xffff5fbff948
Adresa elementului [0][3] din tabloul a este 0xffff5fbff94c
Adresa elementului [0][4] din tabloul a este 0xffff5fbff950
Adresa elementului [1][0] din tabloul a este 0xffff5fbff954
Adresa elementului [1][1] din tabloul a este 0xffff5fbff958
Adresa elementului [1][2] din tabloul a este 0xffff5fbff95c
Adresa elementului [1][3] din tabloul a este 0xffff5fbff960
Adresa elementului [1][4] din tabloul a este 0xffff5fbff964
Adresa elementului [2][0] din tabloul a este 0xffff5fbff968
Adresa elementului [2][1] din tabloul a este 0xffff5fbff96c
Adresa elementului [2][2] din tabloul a este 0xffff5fbff970
Adresa elementului [2][3] din tabloul a este 0xffff5fbff974
Adresa elementului [2][4] din tabloul a este 0xffff5fbff978

3	-12	10	7	1	10	2	0	-7	41	-3	-2	0	0	2
a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[1][0]	...								a[2][4]

Reprezentarea în memoria calculatorului a unui tablou bidimensional

Tablouri bidimensionale

- definiție: set de valori de același tip memorat la adrese succesive de memorie.

- memorie:
 - cantitatea de memorie necesară pentru stocarea unui tablou este direct proporțională cu tipul de date și mărimea sa.
 - tip nume [dimensiune1][dimensiune2] → **sizeof(nume) = sizeof(tip) * dimensiune1 * dimensiune2 ;**

Tablouri bidimensionale

□ memorie:

- cantitatea de memorie necesară pentru stocarea unui tablou este direct proporțională cu tipul de date și mărimea sa.
- tip nume [dimensiune1][dimensiune2] → **sizeof(nume) = sizeof(tip) * dimensiune1 * dimensiune2 ;**

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6
7     double v[100][50];
8     int a[5][25];
9     char c[34][10];
10    printf("Stocarea tabloului v necesita %d octeti \n",sizeof(v));
11    printf("Stocarea tabloului a necesita %d octeti \n",sizeof(a));
12    printf("Stocarea tabloului c necesita %d octeti \n",sizeof(c));
13
14
15 }
```

Stocarea tabloului v necesita 40000 octeti
Stocarea tabloului a necesita 500 octeti
Stocarea tabloului c necesita 340 octeti
Process returned 0 (0x0) execution time : 0.00
Press ENTER to continue.

Tablouri bidimensionale

□ citire și afișare:

The screenshot shows a code editor window titled "main.c". The code is written in C and performs the following tasks:

- #include <stdio.h>
- #include <stdlib.h>
- int main(){
- int L,C,i,j;
- printf("Nr de linii a matricei este L="); scanf("%d",&L);
- printf("Nr de linii a matricei este C="); scanf("%d",&C);
- int a[L][C]; ← valabil in standardul C99
- //CITIRE
- printf("CITIRE matrice a \n");
- for(i=0;i<L;i++)
- for(j=0;j<C;j++)
- {
- printf("a[%d][%d] = ",i,j);
- scanf("%d",&a[i][j]);
- }
- //AFISARE
- printf("AFISARE matrice a \n");
- for(i=0;i<L;i++)
- {
- for(j=0;j<C;j++)
- printf("a[%d][%d] = %d \t ",i,j,a[i][j]);
- printf("\n");
- }
- return 0;

Tablouri bidimensionale

□ citire și afisare:

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6
7     int L,C,i,j;
8     printf("Nr de linii a matricei este L="); scanf("%d",&L);
9     printf("Nr de linii a matricei este C="); scanf("%d",&C);
10    int a[L][C];
11
12    //CITIRE
13    printf("CITIRE matrice a \n");
14    for(i=0;i<L;i++)
15        for(j=0;j<C;j++)
16        {
17            printf("a[%d][%d] = ",i,j);
18            scanf("%d",&a[i][j]);
19        }
20
21    //AFISARE
22    printf("AFISARE matrice a \n");
23    for(i=0;i<L;i++)
24    {
25        for(j=0;j<C;j++)
26            printf("a[%d][%d] = %d \t ",i,j,a[i][j]);
27        printf("\n");
28    }
29    return 0;
30 }
```

Nr de linii a matricei este L=2
Nr de linii a matricei este C=3
CITIRE matrice a
a[0][0] = 1
a[0][1] = 2
a[0][2] = 3
a[1][0] = 4
a[1][1] = 5
a[1][2] = 6
AFISARE matrice a
a[0][0] = 1 a[0][1] = 2 a[0][2] = 3
a[1][0] = 4 a[1][1] = 5 a[1][2] = 6

Process returned 0 (0x0) execution time : 4.999999s
Press ENTER to continue.

Şiruri de caractere

- **un sir de caractere (string)** este o zonă de memorie ocupată cu caractere/char-uri (un char ocupă un octet) terminată cu un octet de valoare 0 (caracterul '\0' are codul ASCII egal cu 0).
- o variabilă care reprezintă un sir de caractere este un pointer (adresa) la primul octet.
- se poate reprezenta ca:
 - tablou de caractere (pointer constant):
 - `char sir1[10];`
 - `char sir2[10] = "exemplu";`
 - pointer la caractere:
 - `char *sir3;`
 - `char *sir4 = "exemplu";`

Codurile ASCII

- cod ASCII = reprezentarea numerică a unui caracter.
(ASCII – American Standard Code for Information Interchange)

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     int i;
6
7     for(i = 33; i <= 127; i++)
8     {
9         printf("%c ",i);
10        if ((i-2) % 10 == 0)
11            printf("\n");
12    }
13    return 0;
14
15 }
16
```

Ce afișează programul?

```
! " # $ % & ' ( ) *
+ , - . / 0 1 2 3 4
5 6 7 8 9 : ; < = >
? @ A B C D E F G H
I J K L M N O P Q R
S T U V W X Y Z [ \
] ^ _ ` a b c d e f
g h i j k l m n o p
q r s t u v w x y z
{ | } ~
```

```
Process returned 0 (0x0)    execution time :
Press ENTER to continue.
```

Codurile ASCII

- cod ASCII = reprezentarea numerică a unui caracter.
(ASCII – American Standard Code for Information Interchange)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	Ø	96	60	140	`	~
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	+	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	,	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	:	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Codurile ASCII

- cod ASCII = reprezentarea numerică a unui caracter.
(ASCII – American Standard Code for Information Interchange)

Extended ASCII Codes

128	Ç	144	É	160	á	176	¤	192	Ł	208	₩	224	α	240	≡
129	ü	145	æ	161	í	177	¤¤	193	ŁŁ	209	〒	225	฿	241	±
130	é	146	Æ	162	ó	178	¤¤¤	194	Ŧ	210	ŦŦ	226	Γ	242	≥
131	â	147	ð	163	ú	179	_	195	ŦŦ	211	₩₩	227	π	243	≤
132	ã	148	ö	164	ñ	180	-	196	-	212	₪	228	Σ	244	ƒ
133	à	149	ò	165	Ñ	181	-	197	+	213	₹	229	σ	245	Ј
134	å	150	û	166	º	182		198	ƒ	214	₹₹	230	μ	246	÷
135	ç	151	ù	167	º	183		199		215	₩₩₩	231	τ	247	≈
136	è	152	ÿ	168	¸	184	_ _	200	ŁŁŁ	216	+	232	Φ	248	°
137	ë	153	Ö	169	Ŕ	185		201	₹₹₹	217	ЈЈЈ	233	ଓ	249	.
138	è	154	Ü	170	Ŕ	186		202	₩₩₩	218	₹₹₹	234	Ѡ	250	.
139	ï	155	¢	171	½	187	_ _	203	₩₩₩	219	■	235	฿	251	√
140	î	156	£	172	¼	188	_ _	204	ƒƒƒ	220	■	236	∞	252	¤
141	ì	157	¥	173	¡	189	_ _	205	=	221	█	237	◊	253	²
142	Ä	158	₱	174	«	190	_ _	206	₣₣₣	222	█	238	€	254	■
143	Å	159	ƒ	175	»	191	_	207	₩₩₩	223	■	239	⌚	255	

Citirea și afișarea sirurilor de caractere

□ citire:

- funcția `scanf` cu modelatorul de format `%s`:
 - atenție: dacă inputul este un sir de caractere cu spațiu citește până la spațiu
- Funcția `fgets`
 - citește și spațiile

□ afișare:

- funcția `printf` cu modelatorul de format `%s`:
- funcția `puts`
 - trece pe linia următoare

Cursul de azi

1. Pointeri
2. Funcții de citire/scriere
3. Tablouri. Siruri de caractere.
4. Structuri, uniuni, câmpuri de biți, enumerări.

Tipuri de date structurate

Limbajul C permite creare tipurilor de date în 5 moduri:

- structura (**struct**) – grupează mai multe variabile sub același nume;
- câmpul de biți (variațiune a structurii) → acces ușor la bitii individuali
- uniunea (**union**) – face posibil ca aceleași zone de memorie să fie definite ca două sau mai multe tipuri diferite de variabile
- enumerarea (**enum**) – listă de constante întregi cu nume
- tipuri definite de utilizator (**typedef**) – definește un nou nume pentru un tip existent

Structuri

- ❑ variabile grupate sub același nume.
- ❑ sintaxa:

```
struct <nume> {  
    < tip 1 >    <variabila 1>;  
    < tip 2 >    <variabila 2>;  
    -----  
    < tip n >    <variabila n>;  
} lista_identificatori_de_tip_struct;
```

- ❑ variabilele care fac parte din structură sunt denumite membri (elemente sau câmpuri) ai structurii.

Structuri

- **struct <nume> {**
 < tip 1 > <variabila 1>;
 < tip 2 > <variabila 2>;

 < tip n > <variabila n>;
} lista_identificatori_de_tip_struct;

□ observații:

- dacă numele structurii (<nume>) lipsește, structura se numește **anonimă**. Dacă lista identificatorilor declarați lipsește, se definește doar tipul structură. Cel puțin una dintre aceste specificații trebuie să existe.
- dacă <nume> este prezent → se pot declara noi variabile de tip structura **struct <nume> <lista noilor identificatori>**;
- referirea unui membru al unei variabile de tip structură → operatorul de selecție punct **.** care precizează identificatorul variabilei și al câmpului.

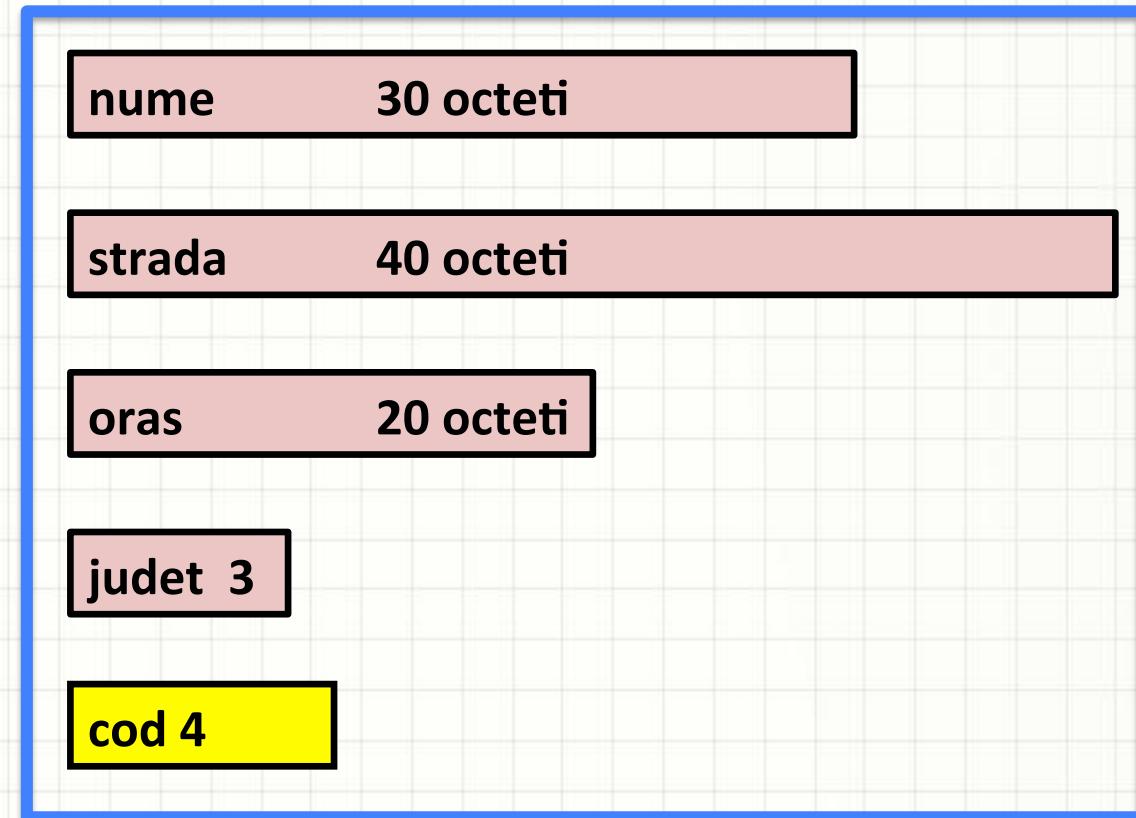
Structuri

- ❑ exemplu:

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20], judet[3];  
    int cod;  
};
```

- ❑ numele **adrese** identifică aceasta structură de date particulară.
- ❑ **struct adrese A;** declară o variabilă de tip adrese și îi aloca memorie

Structura din memorie pentru variabila A
de tip adrese



Structuri

- ❑ exemplu:

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20], judet[3];  
    int cod;  
};
```

The screenshot shows a terminal window with the following content:

```
#include <stdio.h>  
#include <stdlib.h>  
  
int main(){  
    struct adrese{  
        char nume[30];  
        char strada[40];  
        char oras[20], judet[3];  
        int cod;  
    };  
    struct adrese A;  
    printf("A.nume se memorează pe %d octeți \n", (int)sizeof(A.nume));  
    printf("A.cod se memorează pe %d octeți \n", (int)sizeof(A.cod));  
    printf("A se memorează pe %d octeți \n", (int)sizeof(A));  
  
    return 0;  
}
```

The terminal output is:

```
A.nume se memorează pe 30 octeți  
A.cod se memorează pe 4 octeți  
A se memorează pe 100 octeți
```

The line "A se memorează pe 100 octeți" is highlighted with a red box.

Process returned 0 (0x0) execution time : 0.006 s
Press ENTER to continue.

Compilatorul alocă memorie în plus pentru aliniere (multiplu de 4)

Structuri

□ exemplu:

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20], judet[6];  
    int cod;  
};
```

The screenshot shows a code editor with a syntax-highlighted C program. A red box highlights the variable 'judet[6]'. To the right, a terminal window displays the output of the program, which includes memory allocation details for each field.

```
int main(){  
    struct adrese{  
        char nume[30];  
        char strada[40];  
        char oras[20], judet[6];  
        int cod;  
    };  
    struct adrese A;  
    printf("A.nume se memorează pe %d octet", (int)sizeof(A.nume));  
    printf("A.cod se memorează pe %d octeți \n", (int)sizeof(A.cod));  
    printf("A se memorează pe %d octeți \n", (int)sizeof(A));  
  
    return 0;  
}
```

A. nume se memorează pe 30 octeți
A.cod se memorează pe 4 octeți
A se memorează pe 100 octeți
Process returned 0 (0x0)
Press ENTER to continue.

- numele **adrese** identifică aceasta structură de date particulară.
- **struct adrese A;** declară o variabilă de tip adrese și îi aloca memorie

Structuri

- exemplu:

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20], judet[3];  
};
```

The screenshot shows a code editor with the following C code:

```
int main(){  
    struct adrese{  
        char nume[30];  
        char strada[40];  
        char oras[20], judet[3];  
    };  
    struct adrese A;  
    printf("A.nume se memorează pe %d octeți \n", (int)sizeof(A.nume));  
    printf("A se memorează pe %d octeți \n", (int)sizeof(A));  
  
    return 0;  
}
```

A red box highlights the declaration of the `judet` field in the `struct adrese`. Another red box highlights the output of the `printf` statements, which show that the variable `A` occupies 93 bytes of memory.

- numele **adrese** identifică aceasta structură de date particulară.
- **struct adrese A;** declară o variabilă de tip adrese și îi aloca memorie

Compilatorul alocă memorie în plus pentru aliniere numai când avem tipuri de date diferite

Structuri

□ exemplu:

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20], jud[3];  
    int cod;  
} A, B, C;
```

Definește un tip de structură numit **adrese** și declară ca fiind de acest tip variabilele **A, B, C**

```
struct {  
    char nume[30];  
    char strada[40];  
    char oras[20], jud[3];  
    int cod;  
} A;
```

Declară o variabilă numită **A** definită de structura care o precede.

Structuri

- exemplu:
- accesul la membri structurii se face prin folosirea operatorului punct:
nume_variabila.nume_camp
- `scanf("%d", &C.cod);`
- atribuiri pentru variabile de tip structură: **B = A;**

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20], jud[3];  
    int cod;  
} A, B, C;
```

Definește un tip de structură numit **adrese** și declară ca fiind de acest tip variabilele **A, B, C**

Câmpuri de biți

- tip special de membru al unei structuri care definește cât de lung trebuie să fie câmpul, în biți.
- permite accesul la un singur bit.
- putem stoca mai multe variabile boolene într-un singur octet.
- nu se poate obține adresa unui câmp de biți.
- **adaugă mai multă structurare.**

□ **Sintaxa:**

```
struct <nume> {  
    < tip 1 >    <variabila 1>: lungime;  
    < tip 2 >    <variabila 2>: lungime;  
    -----  
    < tip n >    <variabila n>: lungime;  
} lista_identificatori_de_tip_struct;
```

Câmpuri de biți

□ Sintaxa:

```
struct <nume> {
    < tip 1 >  <variabila 1>: lungime;
    < tip 2 >  <variabila 2>: lungime;
    -----
    < tip n >  <variabila n>: lungime;
} lista_identificatori_de_tip_struct;
```

□ Observații:

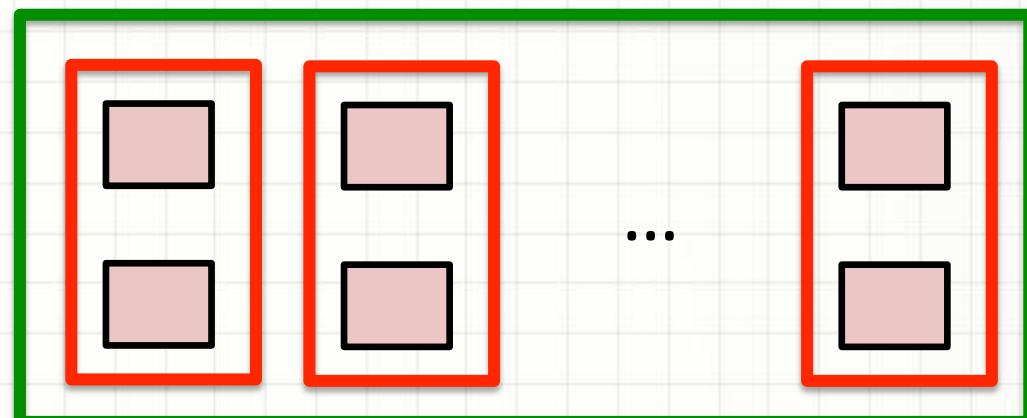
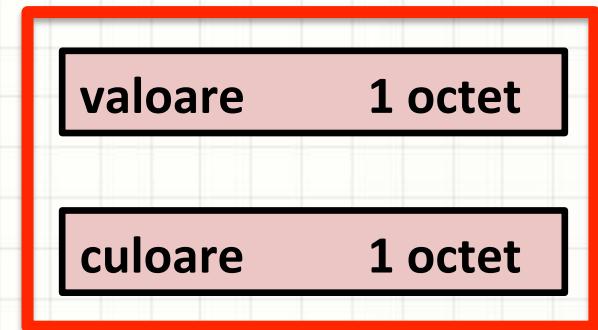
- tipul câmpului de biți poate fi doar: **int**, **unsigned** sau **signed**.
- câmpul de biți cu lungimea 1 → **unsigned** (un singur bit nu poate avea semn).
- unele compilatoare → doar unsigned.
- lungime → numărul de biți dintr-un câmp.

Câmpuri de biți

- exemplu: în cadrul unui joc de cărți reprezentăm o carte printr-o structură

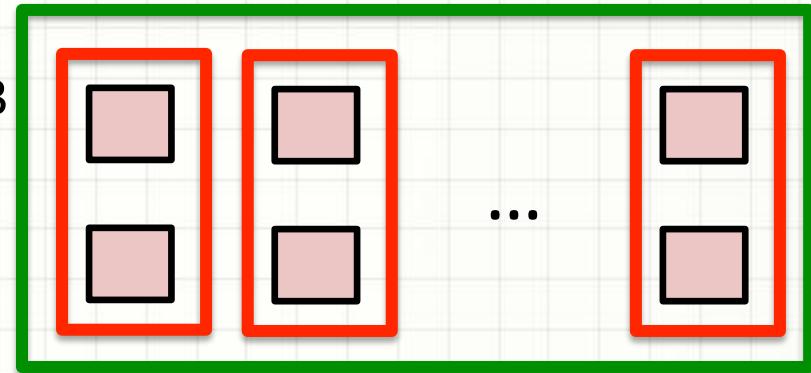
```
struct carteJoc {  
    unsigned char valoare; // valori între 1 și 13  
    unsigned char culoare; // valori între 1 și 4  
};
```

```
struct carteJoc PachetCartiJoc[52];
```



Câmpuri de biți

```
struct carteJoc {  
    unsigned char valoare; // valori între 1 și 13  
    unsigned char culoare; // valori între 1 și 4  
};  
struct carteJoc PachetCartiJoc[52];
```



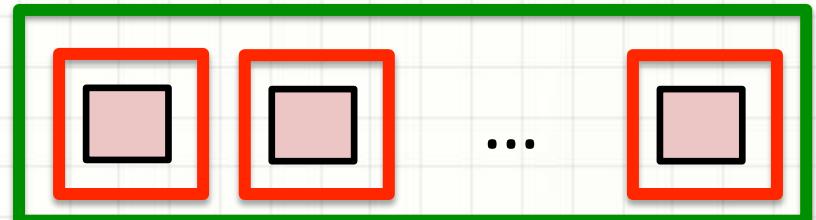
main.c

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3  
4  
5 int main(){  
6     struct carteJoc{  
7         unsigned char valoare;  
8         unsigned char culoare;  
9     } A;  
10  
11     printf("dimensiune A este %d \n", sizeof(A));  
12     struct carteJoc pachetCartiJoc[52];  
13  
14     printf("dimensiune pachetCartiJoc este %d \n", sizeof(pachetCartiJoc));  
15  
16     return 0;  
17 }  
18  
19
```

dimensiune A este 2
dimensiune pachetCartiJoc este 104
Process returned 0 (0x0) execution time : 0
Press ENTER to continue.

Câmpuri de biți

```
struct carteJoc {  
    unsigned char valoare : 4; // 4 biți  
    unsigned char culoare : 2; // 2 biți  
};  
struct carteJoc PachetCartiJoc[52];
```



main.c

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3  
4  
5 int main(){  
6     struct carteJoc{  
7         unsigned char valoare : 4;  
8         unsigned char culoare : 2;  
9     } A;  
10  
11     printf("dimensiune A este %d \n", sizeof(A));  
12     struct carteJoc pachetCartiJoc[52];  
13  
14     printf("dimensiune pachetCartiJoc este %d \n", sizeof(pachetCartiJoc));  
15  
16     return 0;  
17 }  
18  
19
```

dimensiune A este 1
dimensiune pachetCartiJoc este 52
Process returned 0 (0x0) execution time : 0.00
Press ENTER to continue.

Câmpuri de biți

- ❑ exemplu: în aceeași structură putem combina câmpuri de biți și membri normali

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20], jud[3];  
    int cod;  
};
```

```
struct angajat {  
    struct adrese A;  
    float plata;  
    unsigned statut: 1; // activ sau intrerupt  
    unsigned plata_ora: 1; // plata cu ora  
    unsigned impozit: 3; // impozit rezultat  
};
```

- ❑ observație: definește o înregistrare despre salariat care foloseste doar un octet pentru a păstra 3 informații: statutul, daca este platit cu ora și impozitul.
 - ❑ fără câmpul de biti, aceste informații ar fi ocupat 3 octeți.

Uniuni

- ❑ tip special de structuri ai cărei membri folosesc la momente diferite aceeași locație în memorie.
- ❑ membri unei uniuni au de obicei tipuri diferite

Sintaxa:

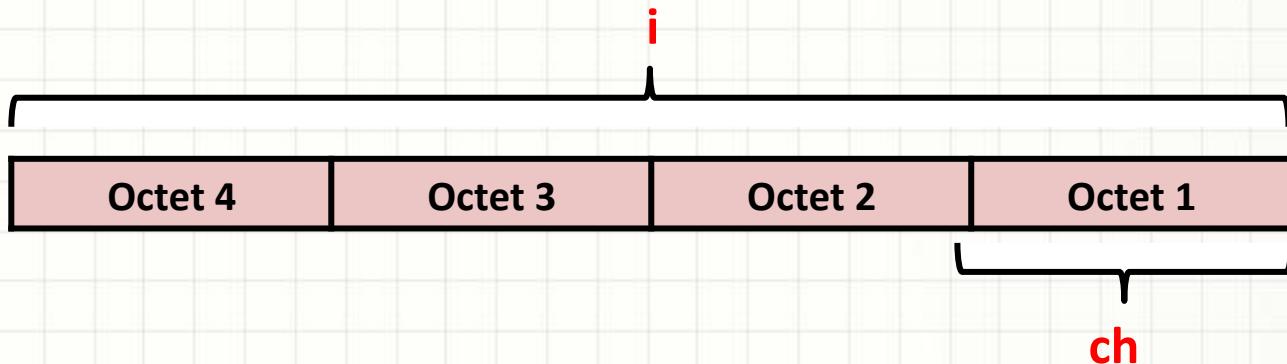
```
union <nume> {  
    < tip 1 >    <variabila 1>;  
    < tip 2 >    <variabila 2>;  
    -----  
    < tip n >    <variabila n>;  
} lista_identificatori_tip_union;
```

Uniuni

- tip special de structuri ai cărei membri folosesc la momente diferite aceeași locație în memorie.
- membri unei uniuni au de obicei tipuri diferite
- exemplu:

```
union tip_u {  
    int i;  
    char ch;  
};
```

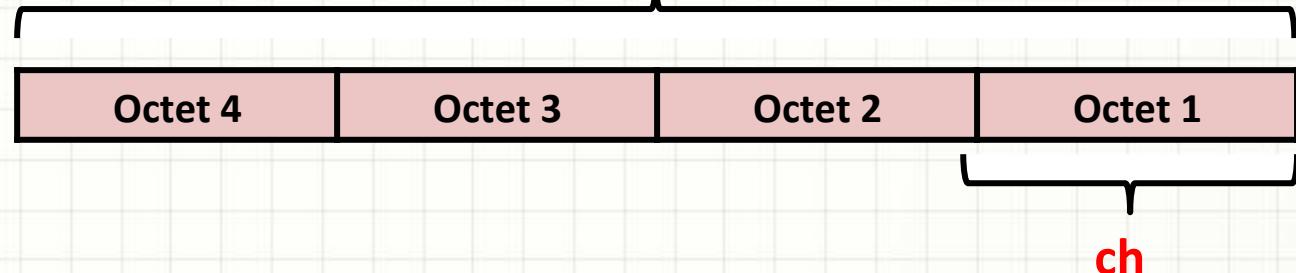
union tip_u A;



- când este declarată o variabilă de tip **union** compilatorul alocă automat memorie suficientă pentru a păstra cel mai mare membru al acesteia.

Uniuni

exemplu



```
main.c X
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6     union tip_u{
7         int i;
8         unsigned char ch;
9     };
10
11     union tip_u A;
12     printf("Dimensiune A = %d \n", sizeof(A));
13     A.ch = 10;
14     printf("A.ch = %d \n", A.ch);
15     printf("A.i = %d \n", A.i);
16
17     A.i = 300;
18     printf("A.ch = %d \n", A.ch);
19     printf("A.i = %d \n", A.i);
20
21
22
23     return 0;
24 }
```

```
Dimensiune A = 4
A.ch = 10
A.i = 10
A.ch = 44
A.i = 300
```

```
Process returned 0 (0x0)    execution time ...
Press ENTER to continue.
```