

# PROGRAMAREA CALCULATOARELOR

Andrei Patrascu

[andrei.patrascu@fmi.unibuc.ro](mailto:andrei.patrascu@fmi.unibuc.ro)

Secția Calculatoare si Tehnologia  
Informatiei, anul I, 2018-2019

Cursul 3

# RECAPITULARE - CURSUL TRECUT

1. Tipuri de date fundamentale
2. Variabile și constante
3. Expresii și operatori

# PROGRAMA CURSULUI

## ❑ Introducere

- Algoritmi
- Limbaje de programare.

## ❑ Fundamentele limbajului C

- Introducere în limbajul C. Structura unui program C.
- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Tipuri derivate de date: tablouri, șiruri de caractere, structuri, uniuni, câmpuri de biți, enumerări, pointeri
- Instrucțiuni de control
- Directive de preprocesare. Macrodefiniții.
- Funcții de citire/scriere.
- Etapele realizării unui program C.

## ❑ Fișiere text

- Funcții specifice de manipulare.

## ❑ Funcții (1)

- Declarare și definire. Apel. Metode de transmitere a paramerilor. Pointeri la funcții.

## ❑ Tablouri și pointeri

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

## ❑ Șiruri de caractere

- Funcții specifice de manipulare.

## ❑ 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.

# CONVERSII IMPLICITE

conversii.c

```
1  #include <stdio.h>
2
3  int main() {
4
5      int i;
6      i = 1.6 + 1 + 1.7;
7      printf("i = %d \n", i);
8      i = (int)1.6 + 1 + (int)1.7;
9      printf("i = %d \n", i);
10
11     char a = 30, b = 40, c = 10;
12     char d = (a*b)/c;
13     printf("%d \n", d);
14
15     unsigned int ui_one = 1;
16     int i_one = 1;
17     short sh_minus_one = -1;
18     if(sh_minus_one > ui_one)
19         printf("-1 > 1 \n");
20     if(sh_minus_one < i_one)
21         printf("-1 < 1 \n");
22
23     return 0;
24 }
```

Ce afișează  
programul  
alăturat?

# CONVERSII IMPLICITE

conversii.c

```
1  #include <stdio.h>
2
3  int main() {
4
5      int i;
6      i = 1.6 + 1 + 1.7;
7      printf("i = %d \n", i);
8      i = (int)1.6 + 1 + (int)1.7;
9      printf("i = %d \n", i);
10
11     char a = 30, b = 40, c = 10;
12     char d = (a*b)/c;
13     printf("%d \n", d);
14
15     unsigned int ui_one = 1;
16     int i_one = 1;
17     short sh_minus_one = -1;
18     if(sh_minus_one > ui_one)
19         printf("-1 > 1 \n");
20     if(sh_minus_one < i_one)
21         printf("-1 < 1 \n");
22
23     return 0;
24 }
```

Ce afișează  
programul  
alăturat?

i = 4

i = 3

# CONVERSII IMPLICITE

conversii.c

```
1  #include <stdio.h>
2
3  int main() {
4
5      int i;
6      i = 1.6 + 1 + 1.7;
7      printf("i = %d \n", i);
8      i = (int)1.6 + 1 + (int)1.7;
9      printf("i = %d \n", i);
10
11     char a = 30, b = 40, c = 10;
12     char d = (a*b)/c;
13     printf("%d \n", d);
14
15     unsigned int ui_one = 1;
16     int i_one = 1;
17     short sh_minus_one = -1;
18     if(sh_minus_one > ui_one)
19         printf("-1 > 1 \n");
20     if(sh_minus_one < i_one)
21         printf("-1 < 1 \n");
22
23     return 0;
24 }
```

Ce afișează  
programul  
alăturat?

i = 4

i = 3

120

# CONVERSII IMPLICITE

conversii.c

```
1  #include <stdio.h>
2
3  int main() {
4
5      int i;
6      i = 1.6 + 1 + 1.7;
7      printf("i = %d \n", i);
8      i = (int)1.6 + 1 + (int)1.7;
9      printf("i = %d \n", i);
10
11     char a = 30, b = 40, c = 10;
12     char d = (a*b)/c;
13     printf("%d \n", d);
14
15     unsigned int ui_one = 1;
16     int i_one = 1;
17     short sh_minus_one = -1;
18     if(sh_minus_one > ui_one)
19         printf("-1 > 1 \n");
20     if(sh_minus_one < i_one)
21         printf("-1 < 1 \n");
22
23     return 0;
24 }
```

Ce afișează  
programul  
alăturat?

i = 4

i = 3

120

-1 > 1

-1 < 1

# CONVERSII IMPLICITE

## □ context

- este permisă **combinarea mai multor operanzi** de tipuri diferite într-o singură expresie

## □ problema

- operatorii binari, care se aplică asupra a doi operanzi, cer ca **tipul operanzilor să fie același** pentru a putea efectua operația

## □ soluție: conversia implicită

- compilatorul **convertește valorile operanzilor la același tip într-un mod transparent** programatorului înaintea generării codului mașină
- există reguli de conversie implicită

## □ alternativă

- conversii explicite: (tip)



# CONVERSII IMPLICITE - REGULI

- când apar într-o expresie tipurile de date **char** și **short** (atât signed și unsigned) sunt convertite la tipul **int** (**promovarea întregilor**)
- În orice operație între două operanzi, ambii operanzi sunt convertiți la tipul de date cel mai înalt în ierarhie

- ierarhia tipurilor de date:  
(nu există **char** și **short**)

- tipul care se reprezintă pe un număr mai mare de octeți are un rang mai mare în ierarhie
- pentru același tip, varianta fără semn are rang mai mare decât cea cu semn
- tipurile reale au rang mai mare decât tipurile întregi

long double  
double  
float  
unsigned long long int  
long long int  
unsigned long int  
long int  
unsigned int  
int



# CONVERSII IMPLICITE - REGULI

## □ conversii implicite la atribuire

- valoarea expresiei din dreapta se convertește la tipul expresiei din stânga
- pot apare pierderi – dacă tipul nu este suficient de încăpător

```
char c = 'a';
short sh = 140;
int a = 3, b;
unsigned int u = 1234567u;
long i = 300L;
float f = 80.13f;
double d = 5.75, g;

b = a + sh; // val. lui sh convertita la int
a = sh - c; // val. lui sh si c convertite la int
g = d + f; // val. lui f convertita la double
f = i + u; // cal lui u convertita la long
           // rezultatul convertit la float
```

# CURSUL DE AZI

1. Pointeri
2. Tablouri. Șiruri de caractere.
3. 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**

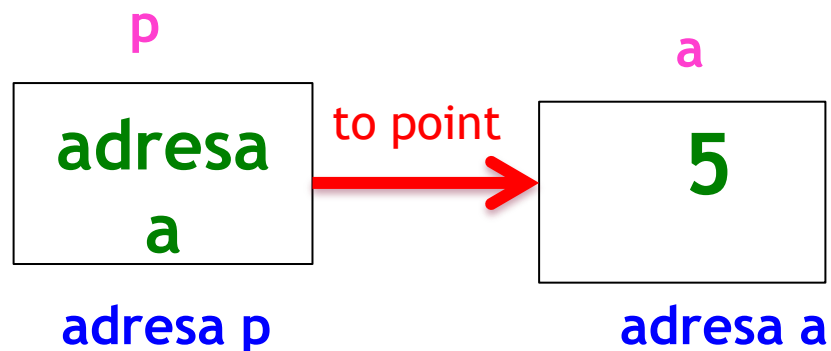
# POINTERI

- **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;
```

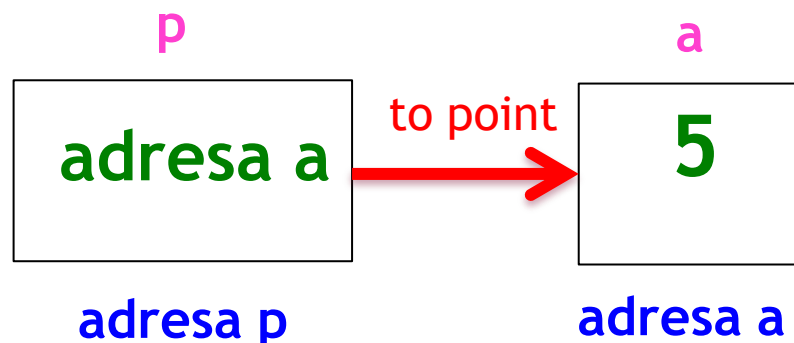


```
main.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      int a=5,*p;
8      p = &a;
9
10     printf("Adresa lui a este: %d \n",&a);
11     printf("Valoarea stocata la adresa lui a este: %d \n",a);
12
13     printf("Adresa lui p este: %d \n",&p);
14     printf("Valoarea stocata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

# POINTERI

❑ **exemplu:**

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



```
main.c x
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("Adresa lui a este: %d \n",&a);
11     printf("Valoarea stocata la adresa lui a este: %d \n",a);
12
13     printf("Adresa lui p este: %d \n",&p);
14     printf("Valoarea stocata la adresa lui p este: %d \n",p);
15     printf("Valoarea variabilei a carei adresa e stocata in p este %d \n",*p);
16
17     return 0;
18 }
19
```

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 x
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
```



# 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 neinițializați 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
```



adresa p

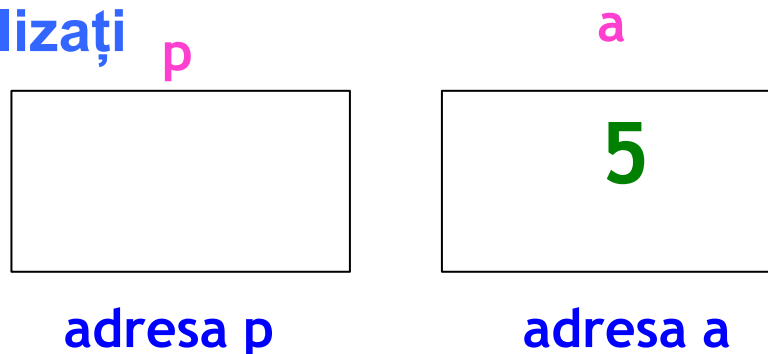


adresa a

# POINTERI

❑ exemplu: pointeri neinițializați

```
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
```



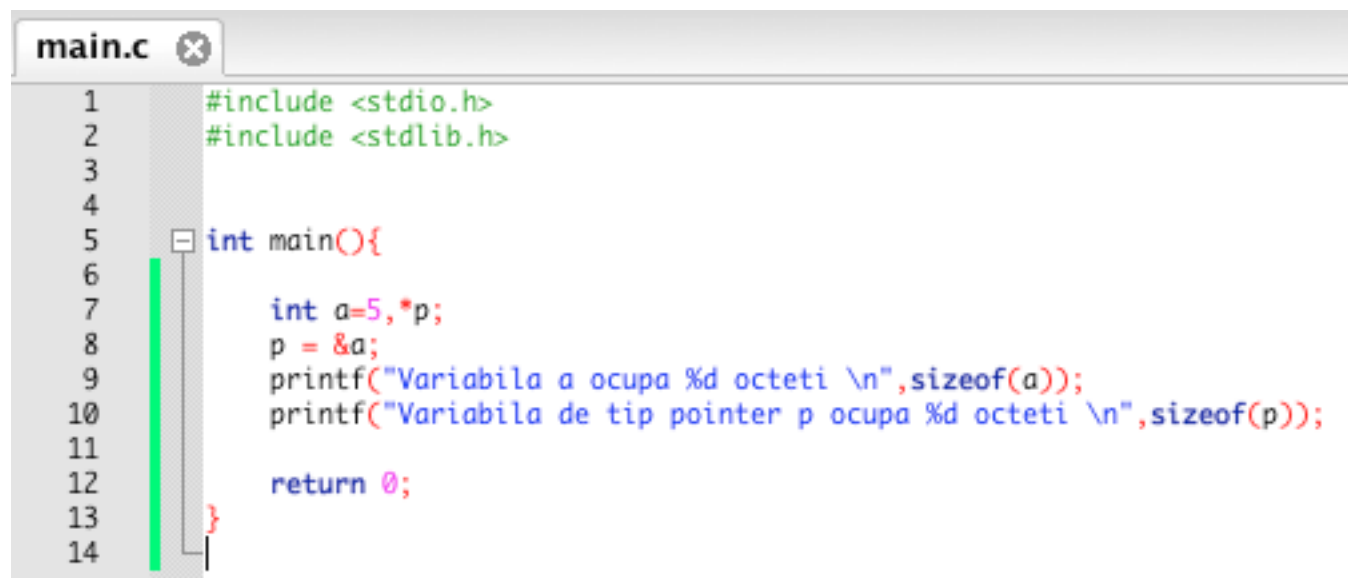
a= 5

Process returned -1 (0xFFFFFFFF)  
Press ENTER to continue.

execution time : 0.027 s

# 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
```

The screenshot shows a code editor window titled 'main.c'. The code is a C program that demonstrates the memory size of an integer variable and a pointer variable. It includes the standard input/output and library headers. The main function declares an integer 'a' with the value 5 and a pointer 'p' that points to 'a'. It then uses 'printf' to output the size of 'a' and 'p' in octets. The program returns 0.

# 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

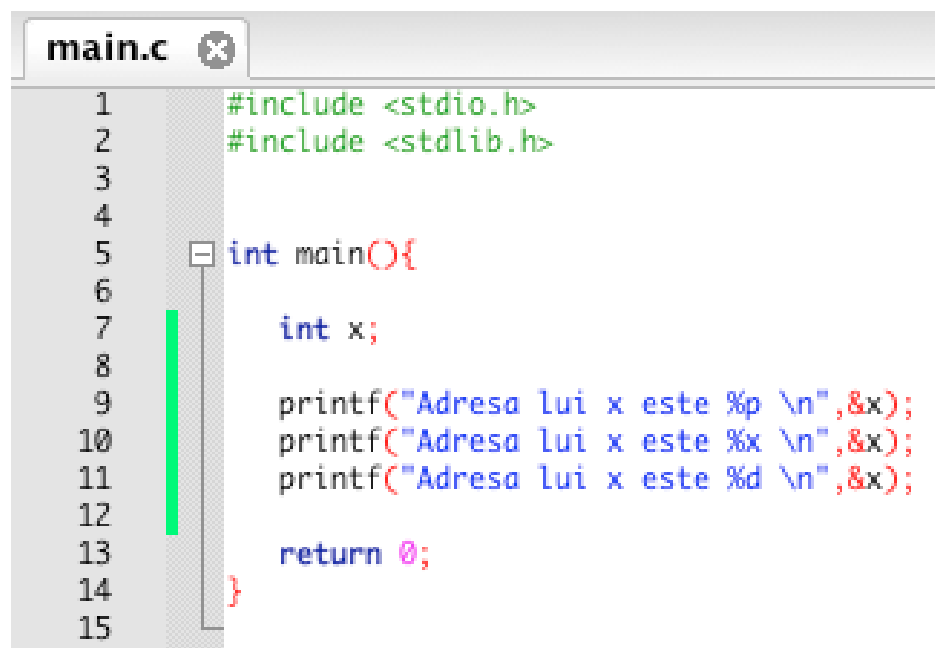
Variabila de tip pointer p ocupa 8 octeti

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 x
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
```

# POINTERI

## ❑ adrese de memorie

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

```
main.c x
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
```

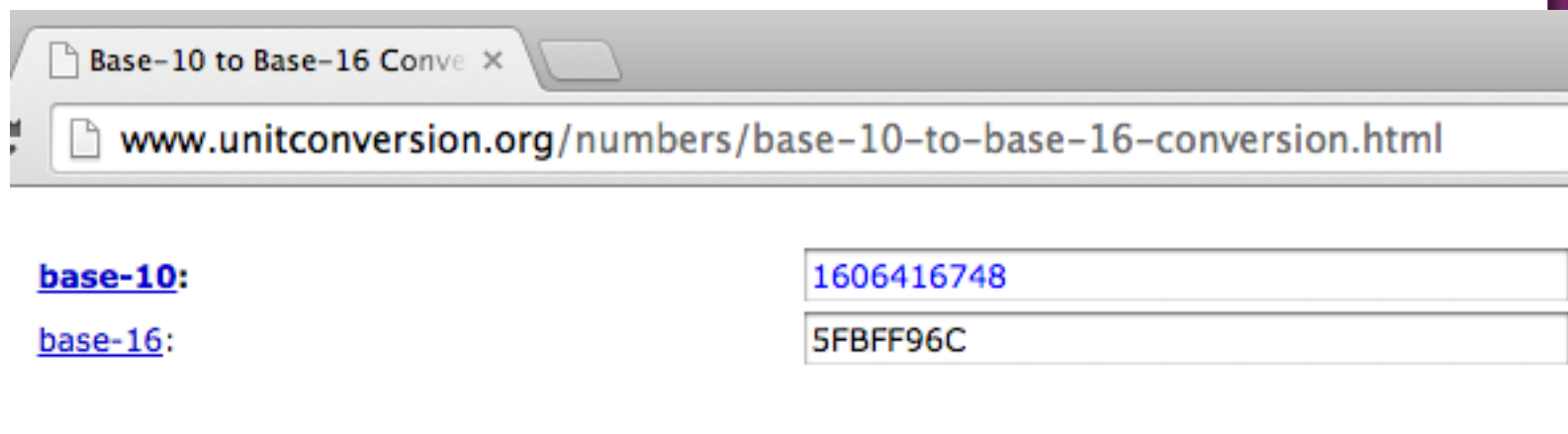
```
Adresa lui x este 0x7fff5fbff96c
Adresa lui x este 5fbff96c
Adresa lui x este 1606416748
```

```
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.



The screenshot shows a web browser window with the title "Base-10 to Base-16 Conve" and the address bar displaying "www.unitconversion.org/numbers/base-10-to-base-16-conversion.html". Below the address bar, there are two input fields. The first field is labeled "base-10:" and contains the value "1606416748". The second field is labeled "base-16:" and contains the value "5FBFF96C".

<u>base-10:</u>	1606416748
<u>base-16:</u>	5FBFF96C



# CURSUL DE AZI

1. Pointeri
2. Tablouri. Șiruri de caractere.
3. Structuri, uniuni, câmpuri de biți, enumerări.

# TABLOURI UNIDIMENSIONALE

## □ sintaxa:

**tip nume\_tablou [dimensiune];**

## □ exemple:

double v[100];

0.3	-1.2	10	5.7	...	0.2	-1.5	1
0	1	2	3		9	9	9
					7	8	9

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		3	3	3
					1	2	3

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.

tabloul.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
```

Adresa elementului 0 din tabloul a este 0x7fff5fbff940  
Adresa elementului 1 din tabloul a este 0x7fff5fbff944  
Adresa elementului 2 din tabloul a este 0x7fff5fbff948  
Adresa elementului 3 din tabloul a este 0x7fff5fbff94c  
Adresa elementului 4 din tabloul a este 0x7fff5fbff950  
\*\*\*\*\*  
Adresa elementului 0 din tabloul v este 0x7fff5fbff620  
Adresa elementului 1 din tabloul v este 0x7fff5fbff628  
Adresa elementului 2 din tabloul v este 0x7fff5fbff630  
\*\*\*\*\*  
Adresa elementului 0 din tabloul c este 0x7fff5fbff960  
Adresa elementului 1 din tabloul c este 0x7fff5fbff961  
Adresa elementului 2 din tabloul c este 0x7fff5fbff962  
Adresa elementului 3 din tabloul c este 0x7fff5fbff963  
\*\*\*\*\*

# 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.
- $\text{tip nume [dimensiune]} \rightarrow \text{sizeof(nume)} = \text{sizeof (tip)} * \text{dimensiune} ;$

```
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     return 0;
13 }
```

# TABLOURI UNIDIMENSIONALE

## □ memorie:

- cantitatea de memorie necesară pentru stocarea unui tablou este direct proporțională cu tipul de date și mărimea sa.
- $\text{tip nume [dimensiune]} \rightarrow \text{sizeof(nume)} = \text{sizeof (tip)} * \text{dimensiune ;}$

```
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     return 0;
13 }
```

```
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 inițializare

```
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 }
```



# TABLOURI UNIDIMENSIONALE

## ▣ exemplu de inițializare

```
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 inițializare

```
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 }
```

# TABLOURI UNIDIMENSIONALE

## ▣ exemplu de inițializare

```
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 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.

```
tablou3.c x
1  #include <stdio.h>
2
3  int main(){
4
5      int i,j;
6      int a[3][5];
7
8      for(i=0;i<3;i++)
9          for(j=0;j<5;j++)
10             printf("Adresa elementului [%d][%d] din tabloul a este %p \n",i,j,&a[i][j]);
11
12     return 0;
13 }
```

# TABLOURI BIDIMENSIONALE

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

```
tablou3.c x
1  #include <stdio.h>
2
3  int main() {
4
5      int i,j;
6      int a[3][5];
7
8      for(i=0;i<3;i++)
9          for(j=0;j<5;j++)
10             printf("Adresa elementului [%d][%d] din tabloul a este %p \n",i,j,&a[i][j]);
11
12     return 0;
13 }
```

Adresa elementului [0][0] din tabloul a este 0x7fff5fbff940  
Adresa elementului [0][1] din tabloul a este 0x7fff5fbff944  
Adresa elementului [0][2] din tabloul a este 0x7fff5fbff948  
Adresa elementului [0][3] din tabloul a este 0x7fff5fbff94c  
Adresa elementului [0][4] din tabloul a este 0x7fff5fbff950  
Adresa elementului [1][0] din tabloul a este 0x7fff5fbff954  
Adresa elementului [1][1] din tabloul a este 0x7fff5fbff958  
Adresa elementului [1][2] din tabloul a este 0x7fff5fbff95c  
Adresa elementului [1][3] din tabloul a este 0x7fff5fbff960  
Adresa elementului [1][4] din tabloul a este 0x7fff5fbff964  
Adresa elementului [2][0] din tabloul a este 0x7fff5fbff968  
Adresa elementului [2][1] din tabloul a este 0x7fff5fbff96c  
Adresa elementului [2][2] din tabloul a este 0x7fff5fbff970  
Adresa elementului [2][3] din tabloul a este 0x7fff5fbff974  
Adresa elementului [2][4] din tabloul a este 0x7fff5fbff978

# 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



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]

```
Adresa elementului [0][0] din tabloul a este 0x7fff5fbff940
Adresa elementului [0][1] din tabloul a este 0x7fff5fbff944
Adresa elementului [0][2] din tabloul a este 0x7fff5fbff948
Adresa elementului [0][3] din tabloul a este 0x7fff5fbff94c
Adresa elementului [0][4] din tabloul a este 0x7fff5fbff950
Adresa elementului [1][0] din tabloul a este 0x7fff5fbff954
Adresa elementului [1][1] din tabloul a este 0x7fff5fbff958
Adresa elementului [1][2] din tabloul a este 0x7fff5fbff95c
Adresa elementului [1][3] din tabloul a este 0x7fff5fbff960
Adresa elementului [1][4] din tabloul a este 0x7fff5fbff964
Adresa elementului [2][0] din tabloul a este 0x7fff5fbff968
Adresa elementului [2][1] din tabloul a este 0x7fff5fbff96c
Adresa elementului [2][2] din tabloul a este 0x7fff5fbff970
Adresa elementului [2][3] din tabloul a este 0x7fff5fbff974
Adresa elementului [2][4] din tabloul a este 0x7fff5fbff978
```

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

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

```
main.c x
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     return 0;
14 }
15
```

# TABLOURI BIDIMENSIONALE

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

```
main.c x
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     return 0;
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:

```
main.c x
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     //CITIRE
12     printf("CITIRE matrice a \n");
13     for(i=0; i<L; i++)
14         for(j=0; j<C; j++)
15         {
16             printf("a[%d][%d] = ", i, j);
17             scanf("%d", &a[i][j]);
18         }
19     //AFISARE
20     printf("AFISARE matrice a \n");
21     for(i=0; i<L; i++)
22     {
23         for(j=0; j<C; j++)
24             printf("a[%d][%d] = %d \t ", i, j, a[i][j]);
25         printf("\n");
26     }
27     return 0;
28 }
```

valabil in  
standardul  
C99

# TABLOURI BIDIMENSIONALE

## citire și afișare:

```
main.c x
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     //CITIRE
12     printf("CITIRE matrice a \n");
13     for(i=0; i<L; i++)
14         for(j=0; j<C; j++)
15         {
16             printf("a[%d][%d] = ", i, j);
17             scanf("%d", &a[i][j]);
18         }
19     //AFISARE
20     printf("AFISARE matrice a \n");
21     for(i=0; i<L; i++)
22     {
23         for(j=0; j<C; j++)
24             printf("a[%d][%d] = %d \t ", i, j, a[i][j]);
25         printf("\n");
26     }
27     return 0;
28 }
```

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 : 4.9

Press ENTER to continue.

# ȘIRURI DE CARACTERE

- ❑ **un șir 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 șir 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
6      int i;
7
8      for(i = 33; i <= 127; i++)
9      {
10         printf("%c ", i);
11         if ((i-2) % 10 == 0)
12             printf("\n");
13     }
14     return 0;
15 }
16
```

Ce afișează programul?

# 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
6      int i;
7
8      for(i = 33; i <= 127; i++)
9      {
10         printf("%c ", i);
11         if ((i-2) % 10 == 0)
12             printf("\n");
13     }
14     return 0;
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)
Press ENTER to continue.
```

execution time :



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



# CODURILE ASCII

❑ **cod ASCII** = reprezentarea numerică a unui caracter.  
(ASCII - American Standard Code for Information

Int **Extended ASCII Codes**

128	Ç	144	É	160	á	176	░	192	Ł	208	⌚	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	ł	209	ƒ	225	β	241	±
130	é	146	Æ	162	ó	178	▓	194	ṽ	210	π	226	Γ	242	≥
131	â	147	ô	163	ú	179		195	ṽ	211	⌚	227	π	243	≤
132	ä	148	ö	164	ñ	180	†	196	—	212	↳	228	Σ	244	∫
133	à	149	ò	165	Ñ	181	‡	197	+	213	ℱ	229	σ	245	∫
134	â	150	û	166	ª	182	‡	198	↳	214	ℱ	230	μ	246	÷
135	ç	151	ù	167	º	183	¶	199	↳	215	‡	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	¶	200	↳	216	‡	232	Φ	248	°
137	ë	153	Ö	169	¬	185	¶	201	ℱ	217	↳	233	⊙	249	.
138	è	154	Ü	170	¬	186	¶	202	↳	218	↳	234	Ω	250	.
139	ï	155	◊	171	½	187	¶	203	¶	219	■	235	δ	251	√
140	î	156	£	172	¼	188	¶	204	↳	220	■	236	∞	252	∞
141	ì	157	¥	173	¡	189	¶	205	=	221	■	237	φ	253	²
142	Ä	158	£	174	«	190	¶	206	↳	222	■	238	ε	254	■
143	Å	159	f	175	»	191	¶	207	↳	223	■	239	∩	255	

# CITIREA ȘI AFIȘAREA ȘIRURILOR DE CARACTERE

## ❑ citire:

- ❑ funcția `scanf` cu modelatorul de format `%s`:
  - ❑ atenție: dacă inputul este un șir 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

# DIFICULTATI LABORATOR

```
unsigned long long int l;  
unsigned char *ch;  
fflush(stdin);  
scanf("%d",&l); //prin citire cu %d se scriu 4 octeti (un int) in l
```

```
printf("sizeof(int) = %d \n\n",sizeof(int));
```

```
ch = &l;  
printf("Valoare data de ultimii 4 octeti: %u \n",l);  
printf("Ultimii 4 octeti: %d %d %d %d\n\n",*(ch+3),*(ch+2),*(ch+1),*ch);  
printf("8 octeti: %d %d %d %d %d %d %d %d\n",*(ch+7),*(ch+6),*(ch+5),*(ch+4),*(ch+3),*(ch+2),*(ch+1),*ch);  
printf("Valoare data de toti cei 8 octeti: %lld \n\n\n",l);
```

```
l = l/10;  
printf("Valoare lui l/10 data de ultimii 4 octeti = %u \n",l);  
printf("Ultimii 4 octeti: %d %d %d %d\n",*(ch+3),*(ch+2),*(ch+1),*ch);  
printf("8 octeti: %d %d %d %d %d %d %d %d\n",*(ch+7),*(ch+6),*(ch+5),*(ch+4),*(ch+3),*(ch+2),*(ch+1),*ch);  
printf("Valoare lui l/10 data de toti cei 8 octeti = %lld \n",l);
```

# DIFICULTATI LABORATOR

```
unsigned long long int l;  
unsigned char *ch;  
fflush(stdin);  
scanf("%d",&l); //prin citire cu %d se scriu 4 octeti (un int) in l
```

```
printf("sizeof(int) = %d \n\n",sizeof(int));
```

```
ch = &l;
```

```
printf("Valoare data de ultimi  
printf("Ultimii 4 octeti: %d %  
printf("8 octeti: %d %d %d %d  
printf("Valoare data de toti c
```

```
l = l/10;
```

```
printf("Valoare lui l/10 data  
printf("Ultimii 4 octeti: %d %  
printf("8 octeti: %d %d %d %d  
printf("Valoare lui l/10 data de toti cei 8 octeti = %lld \n",l);
```

```
234  
sizeof<int> = 4  
Valoare data de ultimii 4 octeti: 234  
Ultimii 4 octeti: 0 0 0 234  
8 octeti: 0 0 0 121 0 0 0 234  
Valoare data de toti cei 8 octeti: 519691043050  
Valoare lui l/10 data de ultimii 4 octeti = 429496753  
Ultimii 4 octeti: 25 153 153 177  
8 octeti: 0 0 0 12 25 153 153 177  
Valoare lui l/10 data de toti cei 8 octeti = 51969104305  
Process returned 0 (0x0) execution time : 1.767 s  
Press any key to continue.
```

# DIFICULTATI LABORATOR

```
unsigned long long int l;  
unsigned char *ch;  
fflush(stdin);  
scanf("%lld",&l); //prin citire cu %d se scriu 4 octeti (un int) in l  
  
printf("sizeof(int) = %d \n\n",sizeof(int));  
  
ch = &l;  
printf("Valoare data de ultimii 4 octeti: %u \n",l);  
printf("Ultimii 4 octeti: %d %d %d %d\n\n",*(ch+3),*(ch+2),*(ch+1),*ch);  
printf("8 octeti: %d %d %d %d %d %d %d %d\n",*(ch+7),*(ch+6),*(ch+5),*(ch+4),*(ch+3),*(ch+2),*(ch+1),*ch);  
printf("Valoare data de toti cei 8 octeti: %lld \n\n\n",l);  
  
l = l/10;  
printf("Valoare lui l/10 data de ultimii 4 octeti = %u \n",l);  
printf("Ultimii 4 octeti: %d %d %d %d\n",*(ch+3),*(ch+2),*(ch+1),*ch);  
printf("8 octeti: %d %d %d %d %d %d %d %d\n",*(ch+7),*(ch+6),*(ch+5),*(ch+4),*(ch+3),*(ch+2),*(ch+1),*ch);  
printf("Valoare lui l/10 data de toti cei 8 octeti = %lld \n",l);
```

# DIFICULTATI LABORATOR

```
unsigned long long int l;
```

```
unsigned char *ch;
```

```
fflush(stdin);
```

```
scanf("%lld",&l); //prin citire cu %d se scriu 4 octeti (un int) in l
```

```
printf("sizeof(int)
```

```
ch = &l;
```

```
printf("Valoare data
```

```
printf("Ultimii 4 oc
```

```
printf("8 octeti: %c
```

```
printf("Valoare data
```

```
l = l/10;
```

```
printf("Valoare lui
```

```
printf("Ultimii 4 octeti: %d %d %d %d\n",*(ch+3),*(ch+2),*(ch+1),*ch);
```

```
printf("8 octeti: %d %d %d %d %d %d %d %d\n",*(ch+7),*(ch+6),*(ch+5),*(ch+4),*(ch+3),*(ch+2),*(ch+1),*ch);
```

```
printf("Valoare lui l/10 data de toti cei 8 octeti = %lld \n",l);
```

```
234
sizeof(int) = 4
```

```
Valoare data de ultimii 4 octeti: 234
```

```
Ultimii 4 octeti: 0 0 0 234
```

```
8 octeti: 0 0 0 0 0 0 0 234
```

```
Valoare data de toti cei 8 octeti: 234
```

```
Valoare lui l/10 data de ultimii 4 octeti = 23
```

```
Ultimii 4 octeti: 0 0 0 23
```

```
8 octeti: 0 0 0 0 0 0 0 23
```

```
Valoare lui l/10 data de toti cei 8 octeti = 23
```

```
Process returned 0 (0x0) execution time : 2.639 s
```

```
Press any key to continue.
```

# DIFICULTATI LABORATOR

- Citire numar sub forma de sir de caractere

```
int nr[10], count = 0;
char ch[10];
scanf("%s", ch);

while (count < strlen(ch))
{
    nr[count] = ch[count] - '0';
    count++;
}
printf("\n\n");

for (count=0 ; count<strlen(ch) ; count++)
    printf(" %d \n", nr[count]);
```

# DIFICULTATI LABORATOR

```
int nr[10], count = 0;
char ch[10];
scanf("%s", ch);

while (count < strlen(ch))
{
    nr[count] = ch[count] - '0';
    count++;
}
printf("\n\n");

for (count=0 ; count<10 ; count++)
    printf(" %d", nr[count]);
```

123

1  
2  
3

Process returned 0 (0x0) execution time : 4.372 s  
Press any key to continue.



