

PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,

2018-2019

Cursul 2

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, siruri 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)

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

□ Tablouri și pointeri

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

□ Siruri 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.
- Programare generică.

□ Recursivitate

Cuprinsul cursului de azi

1. Introducere în limbajul C. Structura unui program C
2. Tipuri de date fundamentale
3. Variabile și constante
4. Expresii și operatori

Limbajul C

- ❑ popular, rapid și independent de platformă
- ❑ este un limbaj utilizat cel mai adesea pentru scrierea programelor eficiente și portabile: sisteme de operare, aplicații embedded, compilatoare, interpretoare, etc.
- ❑ limbajul C a fost dezvoltat la începutul anilor 1970 în cadrul Bell Laboratories de către Dennis Ritchie
 - ❑ strâns legat de sistemele de operare UNIX
- ❑ stă la baza pentru majoritatea limbajelor "moderne": C++, Java, C#, Javascript, Objective-C, etc.

Limbajul C

- ❑ trei **standarde oficiale active ale limbajului**
 - ❑ **C89** (C90) – aprobat în 1989 de ANSI (American National Standards Institute) și în 1990 de către ISO (International Organization for Standardization)
 - ❑ C89 a eliminat multe din incertitudinile legate de sintaxa limbajului
 - ❑ cele mai multe compilatoare de C sunt compatibile cu acest standard (ANSI C)
 - ❑ **C99** – standard aprobat în 1999, care include corecturile aduse C89 dar și o serie de caracteristici proprii care în unele compilatoare apăreau ca extensii ale C89 până atunci
http://www.dii.uchile.cl/~daespino/files/Iso_C_1999_definition.pdf
 - ❑ **C11** – standard aprobat în 2011 și care rezolvă erorile apărute în standardul C99 și introduce noi elemente, însă suportul pentru C11 este limitat, majoritatea compilatoarelor nu s-au adaptat încă la acest standard

Caracteristici ale limbajului C

- ❑ limbaj procedural, structurat, compilat, de nivel de mijloc, scurt
- ❑ limbaj procedural, structurat
 - ❑ instrucțiuni specificate sub forma unor comenzi grupate într-o ierarhie de subprograme (denumite funcții) și care pot forma module
 - ❑ orice algoritm poate fi compus numai din trei structuri de calcul: secvențială, decizională, repetitivă
- ❑ limbaj compilat
 - ❑ compilatorul transformă instrucțiunile în C din fișierul sursă în limbaj mașină
- ❑ limbaj de nivel de mijloc
 - ❑ permite accesul la date aflate aproape de nivelul fizic folosind o sintaxă specifică limbajelor de nivel înalt
- ❑ limbaj scurt
 - ❑ număr redus de cuvinte cheie
 - ❑ funcționalități neincluse în limbaj - includerea de biblioteci standard

Caracteristici ale limbajului C

- limbaj eficient, portabil, permisiv**, poate fi **dificil de înțeles**
- limbaj eficient**
 - viteză mare de execuție a programelor, destinat și aplicațiilor implementate în limbaj de asamblare
 - reutilizarea ulterioară a subprogramelor
- limbaj portabil**
 - limbaj independent de hardware
- limbaj permisiv**
 - impune puține constrângeri, dă credit programatorului
 - permite introducerea unor erori care sunt foarte greu de depistat
- limbaj dificil de înțeles**
 - un stil de programare adecvat este foarte important
 - obfuscated C code contest: www.ioccc.org

```
#include      <stdio.h>
#define TA      q/*XYXY*/
#define/*X      YXY*/CG r=
void p(int   n,int c){;
for(;n--;)  putchar(c)
#define Y(      z)d;d=c++\
%2<1?x=x*4  +z,c%8>5?\n
x=x?p(1,x), 0:x:0:0;d=
#define/*X      YX*/C Y(1)
#define/*X      YX*/G Y(2)
;int(*f)( void),d,c,
#define/*X      YX*/A Y(0)
#define/*XY*/AT int\m(void/**/){d=
#define/*XYX*/T Y(3)
#define GC d; return\0; }int(*f)( void )=m;
x,q,r; int main(){if(f)f();else{for(puts(
"#include" "\40\"pro\g.c\"\n\n\101T"+0);
d!=d?x=(x=getchar())
<0?0:x,8*8 :d,TA++c%8
,TA(1+7*q-q*q)/3,r=c
*15-c*c-36 ,p(r<0?!q+
4:r/6+!q+4 ,32),q||x;
c%16)q?p( 1,"ACGT"[x/d&3]),p(q
,TGCA"[x/d&3]),d/=4,
p(001,10): puts(c%8?\n"CG":"TA")
;};}return 0; }/**/
```

```
1 #include      <stdio.h>
2 #define TA      q/*XYXY*/
3 #define/*X      YXY*/CG r=
4 void p(int   n,int c){;
5 for(;n--;)  putchar(c)
6 #define Y(      z)d;d=c++\
7 %2<1?x=x*4  +z,c%8>5?\n
8 x=x?p(1,x), 0:x:0:0;d=
9 #define/*X      YX*/C Y(1)
10 #define/*X      YX*/G Y(2)
11 ;int(*f)( void),d,c,
12 #define/*X      YX*/A Y(0)
13 #define/*XY*/AT int\m(void/**/){d=
14 #define/*XYX*/T Y(3)
15 #define GC d; return\0; }int(*f)( void )=m;
16 x,q,r; int main(){if(f)f();else{for(puts(
17 "#include" "\40\"pro\g.c\"\n\n\101T"+0);
18 d!=d?x=(x=getchar())
19 <0?0:x,8*8 :d,TA++c%8
20 ,TA(1+7*q-q*q)/3,r=c
21 *15-c*c-36 ,p(r<0?!q+
22 4:r/6+!q+4 ,32),q||x;
23 c%16)q?p( 1,"ACGT"[x/d&3]),p(q
24 ,TGCA"[x/d&3]),d/=4,
25 p(001,10): puts(c%8?\n"CG":"TA")
26 ;};}return 0; }/**/
```


Cuvinte cheie

C89 = ANSI C : 32 de cuvinte cheie

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

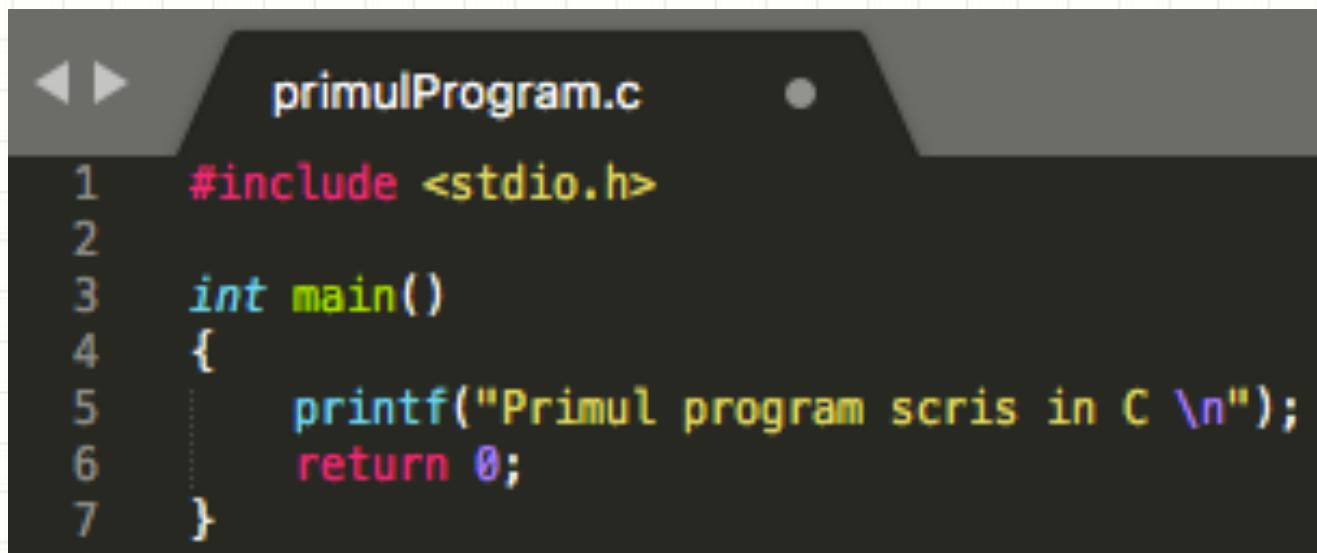
C99: ANSI C + alte 5 cuvinte cheie

_Bool _Complex _Imaginary inline restrict

Structura generală a unui program C

- modul principal (funcția main)
- zero, unul sau mai multe module (funcții/proceduri) care comunică între ele și/sau cu modulul principal prin intermediul parametrilor și/sau a unor variabile globale
- unitatea de program cea mai mică și care conține cod este funcția/procedura și conține:
 - partea de declarații/definiții;
 - partea imperativă (comenzile care se vor executa);

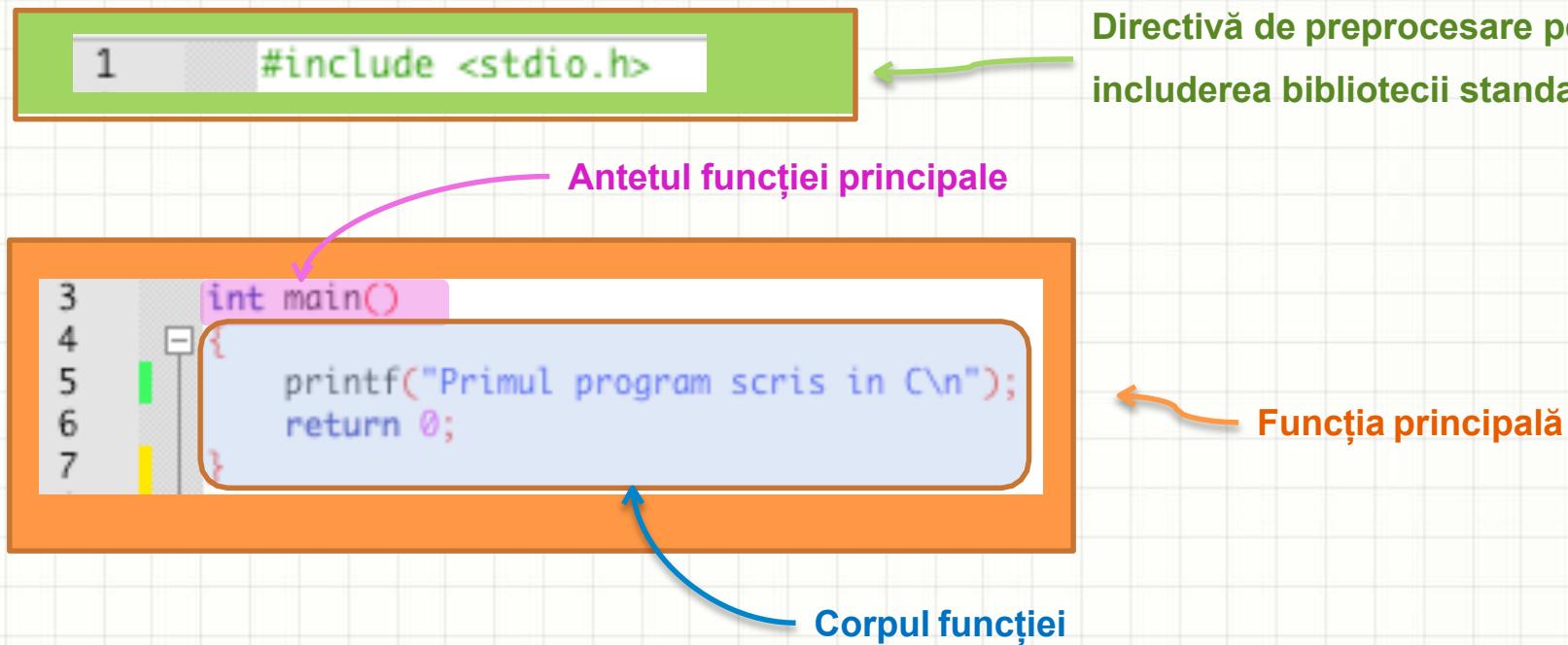
Primul program în C



A screenshot of a code editor window titled "primulProgram.c". The code editor has a dark theme with syntax highlighting. The code itself is a simple C program:

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Primul program scris in C \n");
6     return 0;
7 }
```

Primul program în C



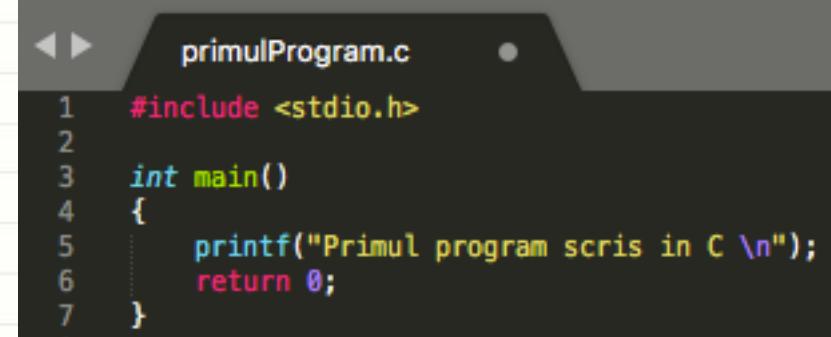
Observații:

- `main` nu este cuvânt cheie în limbajul C, îl utilizăm pentru numirea funcției principale;
- `printf` nu este cuvânt cheie, este funcție de bibliotecă (print (afişare) +f (format));
- C este case sensitive, se face diferență între litere mici și mari;
- toate cuvintele cheie se scriu cu litere mici;
- instrucțiunile se termină cu `caracterul ;` (punct și virgulă);
- mai multe instrucțiuni pot fi scrise pe aceeași linie;
- spațiile ajută la organizarea codului.

Structura unui program C simplu

directive de procesare

```
int main()
{
    instrucțiuni
}
```



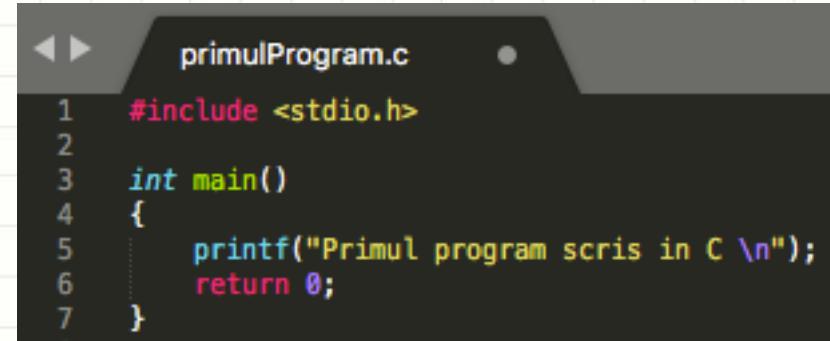
```
primulProgram.c
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Primul program scris in C \n");
6     return 0;
7 }
```

- directive de procesare
 - directive de definiție: #define N 10
 - directive de includere a bibliotecilor: #include <stdio.h>
 - directive de compilare condiționată: #if, #ifdef, ...
 - alte directive (vorbim în cursurile următoare)
- funcții
 - grupări de instrucțiuni sub un nume;
 - returnează o valoare sau se rezumă la efectul produs;
 - funcții scrise de programator vs. funcții furnizate de biblioteci;
 - programul poate conține mai multe funcții;
 - **main** este obligatoriu;
 - antetul și corpul funcției.

Structura unui program C simplu

directive de preprocessare

```
int main()
{
    instrucțiuni
}
```



```
primulProgram.c
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Primul program scris in C \n");
6     return 0;
7 }
```

□ instrucțiuni

- formează corpul funcțiilor
 - exprimate sub formă de comenzi
- 5 tipuri de instrucțiuni:
 - instrucțiunea declarație;
 - instrucțiunea atribuire;
 - instrucțiunea apel de funcție;
 - instrucțiuni de control;
 - instrucțiunea vidă;
- toate instrucțiunile (cu excepția celor compuse) se termină cu caracterul ";"
 - caracterul ; nu are doar rol de separator de instrucțiuni ci instrucțiunile încorporează caracterul ; ca ultim caracter
 - omiterea caracterului ; reprezintă eroare de sintaxă

Structura unui program C complex

comentarii

directive de preprocessare

declarații și definiții globale

```
int main()
```

```
{
```

declarații și definiții locale

instrucțiuni

```
}
```

Cuprinsul cursului de azi

1. Introducere în limbajul C. Structura unui program C
2. Tipuri de date fundamentale
3. Variabile și constante
4. Expresii și operatori

Tipuri de date fundamentale

- ❑ programele manipulează date sub formă de numere, litere, cuvinte, etc.
- ❑ tipul de date specifică:
 - ❑ natura datelor care pot fi stocate în variabilele de acel tip
 - ❑ necesarul de memorie
 - ❑ operațiile permise asupra acestor variabile
- ❑ În C89, limbajul C are **cinci categorii fundamentale** de tipuri de date: **int, char, double, float, void**
- ❑ C99 a introdus alte 3 tipurile de date:
 - ❑ **_Bool** (true, false), de fapt valori întregi (0 = fals, altceva = adevărat)
 - ❑ **_Complex** pentru numere complexe
 - ❑ **_Imaginary** pentru numere imaginare

Tipuri de date fundamentale

- În C89, limbajul C are **cinci categorii fundamentale** de tipuri de date: **int**, **char**, **double**, **float**, **void**
 - tipul **întreg** – **int**: variabilele de acest tip pot reține valori întregi ca 2, 0, -532
 - tipul **caracter** – **char**: variabilele de acest tip pot reține codul ASCII al unui caracter (număr întreg) sau numere întregi mici
 - tipul **real** (numere în virgulă mobilă) – **simplă precizie** – **float**: variabilele de acest tip pot reține numere care conțin parte fraționară: 4971.185, -0.72561, 2.000, 3.14
 - tipul **real** (numere în virgulă mobilă) **în dublă precizie** – **double**: variabilele de acest tip pot reține valori reale în virgulă mobilă cu o precizie mai mare decât tipul float
 - tipul **void**: indică lipsa unui tip anume

Tipuri de date fundamentale

- se pot crea noi tipuri de date prin combinarea celor de bază
- reprezentarea și spațiul ocupat în memorie de diferitele tipuri de date depind de:
 - platformă, sistem de operare și compilator
- limitele specifice unui sistem de calcul pot fi aflate din fișierele header `limits.h` și `float.h`
 - exemplu: `CHAR_MAX`, `INT_MAX`, `INT_MIN`, `FLT_MAX`, `DBL_MAX`
- pentru determinarea numărului de octeți ocupați de un anumit tip de date se folosește operatorul `sizeof`
 - `1 octet = 1 byte = 8 biți`

Spațiul ocupat în memorie

```
dimensiuneOcteti.c  x

1 #include <stdio.h>
2 #include <limits.h>
3 #include <float.h>
4
5 int main()
6 {
7     //tipul char
8     printf("\nsizeof(char) = %lu \n", sizeof(char));
9     printf("valoarea minima pt o variabila de tip char este %d \n", CHAR_MIN);
10    printf("valoarea maxima pt o variabila de tip char este %d \n\n", CHAR_MAX);
11
12    //tipul int
13    printf("sizeof(int) = %lu \n", sizeof(int));
14    printf("valoarea minima pt o variabila de tip int este %d \n", INT_MIN);
15    printf("valoarea maxima pt o variabila de tip int este %d \n\n", INT_MAX);
16
17    //tipul float
18    printf("sizeof(float) = %lu \n", sizeof(float));
19    printf("valoarea minima > 0 pt o variabila de tip float este %E \n", FLT_MIN);
20    printf("valoarea maxima pt o variabila de tip float este %E \n", FLT_MAX);
21    printf("valoarea maxima pt o variabila de tip float este %lf \n", FLT_MAX);
22    printf("Precizia folosita pentru variabile de tip float este de %d zecimale\n\n\n", FLT_DIG);
23
24    //tipul double
25    printf("sizeof(double) = %lu \n", sizeof(double));
26    printf("valoarea minima > 0 pt o variabila de tip double este %E \n", DBL_MIN);
27    printf("valoarea maxima pt o variabila de tip double este %E \n", DBL_MAX);
28    printf("valoarea maxima pt o variabila de tip double este %lf \n", DBL_MAX);
29    printf("Precizia folosita pentru variabile de tip double este de %d zecimale\n\n\n", DBL_DIG);
30
31    return 0;
32 }
```

Spațiul ocupat în memorie

```
sizeof(char) = 1
valoarea minima pt o variabila de tip char este -128
valoarea maxima pt o variabila de tip char este 127

sizeof(int) = 4
valoarea minima pt o variabila de tip int este -2147483648
valoarea maxima pt o variabila de tip int este 2147483647

sizeof(float) = 4
valoarea minima > 0 pt o variabila de tip float este 1.175494E-38
valoarea maxima pt o variabila de tip float este 3.402823E+38
valoarea maxima pt o variabila de tip float este 340282346638528859811704183484516925440.000000
Precizia folosita pentru variabile de tip float este de 6 zecimale

sizeof(double) = 8
valoarea minima > 0 pt o variabila de tip double este 2.225074E-308
valoarea maxima pt o variabila de tip double este 1.797693E+308
valoarea maxima pt o variabila de tip double este 1797693134862315708145274237317043567980705675
258449965989174768031572607800285387605895586327668781715404589535143824642343213268894641827684
675467035375169860499105765512820762454900903893289440758685084551339423045832369032229481658085
59332123348274797826204144723168738177180919299881250404026184124858368.000000
Precizia folosita pentru variabile de tip double este de 15 zecimale
```

Importanța tipurilor de date



java binary search bug



All

Images

Videos

News

Maps

More

Settings

Tools

About 3,170,000 results (0.47 seconds)

Nearly All Binary Searches and Mergesorts are Broken - Google AI Blog

<https://ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html> ▾

Jun 2, 2006 - So what's the **bug**? Here's a standard **binary search**, in Java. (It's one that I wrote for the `java.util.Arrays`): 1: public static int **binarySearch**(int[] a, ...

There was a bug in Java's `Arrays.binarySearch()`, that was ...

<https://www.quora.com/There-was-a-bug-in-Javas-Arrays-binarySearch-that-was-disc...> ▾

Oct 26, 2014 - In short, an integer overflow **bug** when calculating the midpoint of the range that you're dividing the **search** over. This **bug** is surprisingly prevalent; even Jon Bentley of Programming Pearls fame fell prey to it, and I know I did too more than once in my college days.

What are common mistakes for implementing **binary search**? - Quora Jul 19, 2015

First **Binary Search** was published in 1946; first **bug** free one in ... Mar 18, 2015

More results from www.quora.com

The curious case of Binary Search — The famous bug that remained ...

<https://thebittheories.com/the-curious-case-of-binary-search-the-famous-bug-that-rem...> ▾

Jan 4, 2018 - All Divide and Conquer algorithms (like **Binary Search**, Merge Sort) are vulnerable to this simple **bug** that went undetected for decades. Lets understand how to fix it ... Even Java's **Binary search** in `java.util.Arrays` had the same ...

So what's the bug? Here's a standard binary search, in Java. (It's one that I wrote for the `java.util.Arrays`):

```
1:  public static int binarySearch(int[] a, int key) {  
2:      int low = 0;  
3:      int high = a.length - 1;  
4:  
5:      while (low <= high) {  
6:          int mid = (low + high) / 2;  
7:          int midVal = a[mid];  
8:  
9:          if (midVal < key)  
10:              low = mid + 1  
11:          else if (midVal > key)  
12:              high = mid - 1;  
13:          else  
14:              return mid; // key found  
15:      }  
16:      return -(low + 1); // key not found.  
17:  }
```

The bug is in this line:

```
6:         int mid = (low + high) / 2;
```

In *Programming Pearls* Bentley says that the analogous line "sets m to the average of l and u, truncated down to the nearest integer." On the face of it, this assertion might appear correct, but it fails for large values of the `int` variables `low` and `high`. Specifically, it fails if the sum of `low` and `high` is greater than the maximum positive `int` value ($2^{31} - 1$). The sum overflows to a negative value, and the value stays negative when divided by two. In C this causes an array index out of bounds with unpredictable results. In Java, it throws `ArrayIndexOutOfBoundsException`.

This bug can manifest itself for arrays whose length (in elements) is 2^{30} or greater (roughly a billion elements). This was inconceivable back in the '80s, when *Programming Pearls* was written, but it is common these days at Google and other places. In *Programming Pearls*, Bentley says "While the first binary search was published in 1946, the first binary search that works correctly for all values of n did not appear until 1962." The truth is, very few correct versions have ever been published, at least in mainstream programming languages.

So what's the best way to fix the bug? Here's one way:

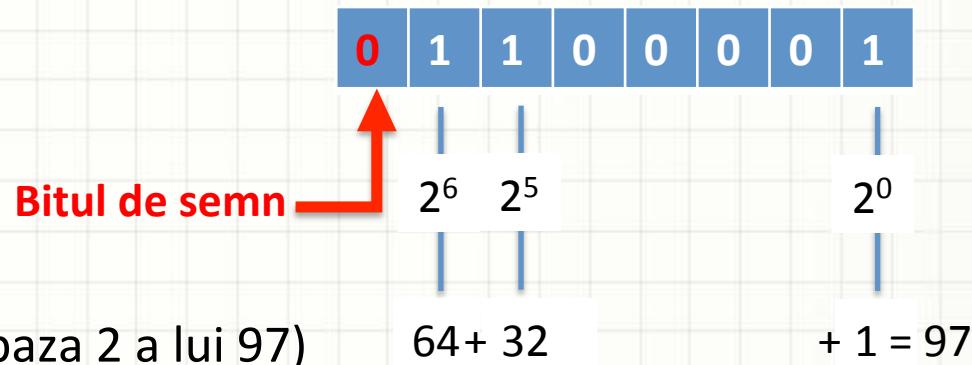
```
6:         int mid = low + ((high - low) / 2);
```

Reprezentarea în memorie

- **char**: ocupă 1 octet = 8 biți, valori între $-2^7 = -128$ și $2^7 - 1 = 127$

`char ch = 'a';`

'a' are codul ASCII 97



$$97 = 2^6 + 2^5 + 2^0 \text{ (scrierea în baza 2 a lui 97)}$$

- **int**: ocupă 4 octeți = 32 biți, valori între -2^{31} și $2^{31} - 1$

`int i = 190;`



Reprezentarea binara a lui 190 in memoria calculatorului

$$190 = 2^7 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 \text{ (scrierea în baza 2 a lui 190)}$$

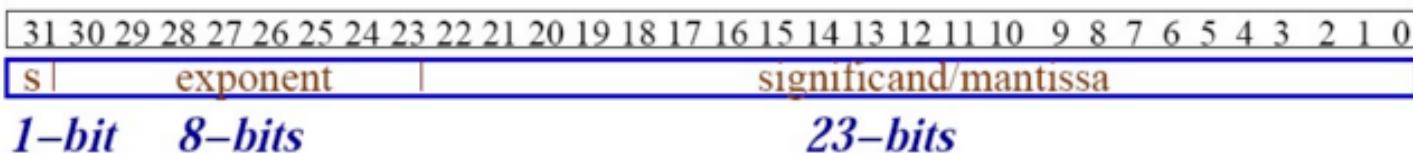
Big-Endian (cel mai semnificativ octet este memorat primul) versus Little-Endian (cel mai puțin semnificativ octet este memorat primul)

Reprezentarea în memorie



$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent-Bias})}$$

- float: ocupă 4 octeți = 32 biți, precizie simplă, bias = 127



float f = 7.0; $7.0 = 1.75 * 4 = (-1)^0 * (1+0.75) * 2^{(129-127)}$

Reprezentare binara exponent: $129 = 128 + 1 = 2^7 + 2^0$

Reprezentare binara fractie: $0.75 = 0.5 + 0.25 = 2^{-1} + 2^{-2}$



Reprezentarea binara a lui 7.0 (float) in memoria calculatorului

Reprezentarea în memorie

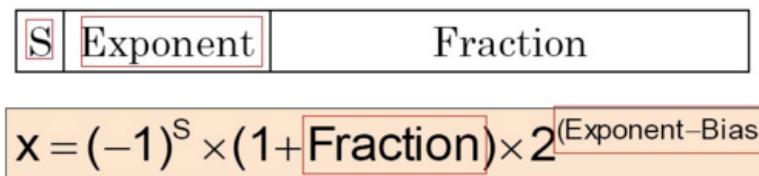
- multimea numerelor reale care pot fi reprezentate de variabile de tip float nu este repartizată uniform
- aproape jumătate din ele sunt în intervalul [-1, 1]

S	Exponent	Fraction
$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent-Bias})}$		

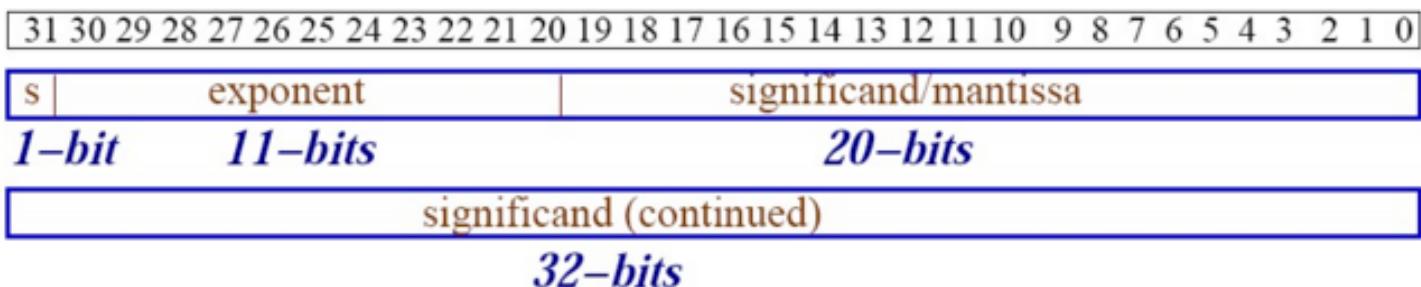
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	exponent																									significand/mantissa					
	1-bit	8-bits										23-bits																			

- pentru $s = 0$ și $1 \leq \text{exponent} < 127$ obținem un număr pozitiv subunitar. Există $126 * 2^{23}$ asemenea numere reprezentabile de variabile de tip float. Există 2^{32} numere reale reprezentabile de tip float.
- $126 * 2^{23} / 2^{32} = 126 / 512 \approx 24,6\%$ nr pozitive subunitare
- la fel pentru $s = 1$
- 49,2% din numere reprezentate de float sunt în [-1, 1]

Reprezentarea în memorie



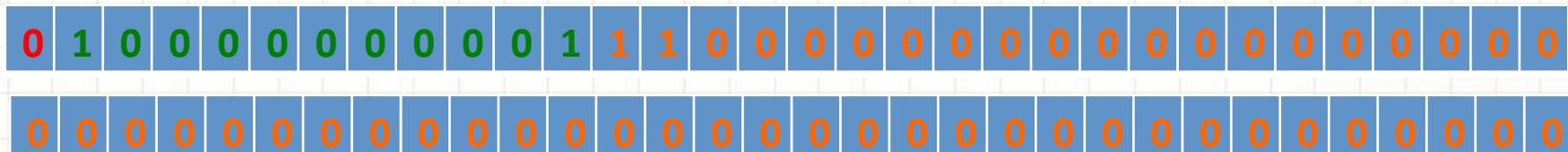
- double**: ocupă 8 octeți = 64 biți, precizie dublă, bias = 1023



double d = 7.0; $7.0 = 1.75 * 4 = (-1)^0 * (1+0.75) * 2^{(1025-1023)}$

Reprezentare binara exponent: $1025 = 1024 + 1 = 2^{10} + 2^0$

Reprezentare binara fractie: $0.75 = 0.5 + 0.25 = 2^{-1} + 2^{-2}$



Există 10 tipuri de
studenți la FMI:
cei care înțeleg sistemul
binar și cei care nu îl
înțeleg

Modificatori de tip

- **signed**
 - modificatorul implicit pentru toate tipurile de date
 - bitul cel mai semnificativ din reprezentarea valorii este semnul
- **unsigned**
 - restricționează valorile numerice memorate la valori pozitive
 - domeniul de valori este mai mare deoarece bitul de semn este liber și participă în reprezentarea valorilor
- **short**
 - reduce dimensiunea tipului de date întreg la jumătate
 - se aplică doar pe întregi
- **long**
 - permite memorarea valorilor care depășesc limita specifică tipului de date
 - se aplică doar pe int sau double: la int dimensiunea tipului de bază se dublează, la double se mărește dimensiunea de regulă cu doi octeți (de la 8 la 10 octeți)
- **long long**
 - introdus în C99 pentru stocarea unor valori întregi de dimensiuni foarte mari

Tipuri de date + modificatori

Tip de date + modificator	Dimensiune în biți	Domeniu de valori
char	8	de la -128 la 127
unsigned char	8	de la 0 la 255
signed char	8	de la -128 la 127
int	32	de la -2^{31} la $2^{31}-1$
unsigned int	32	de la 0 la $2^{32}-1$
signed int	32	de la -2^{31} la $2^{31}-1$
short int	16	de la -2^{15} la $2^{15}-1$
unsigned short int	16	de la 0 la $2^{16}-1$
signed short int	16	de la -2^{15} la $2^{15}-1$
long int	64	de la -2^{63} la $2^{63}-1$
float	32	...
double	64	...

Signed vs. unsigned

- ❑ **signed int** – întreg cu semn (pozitiv sau negativ)
- ❑ **unsigned int** – întreg fără semn (pozitiv)

```
int i = 190;
```



A horizontal bar divided into 32 equal segments, each representing a bit of a 32-bit integer. The first segment is red and contains the digit '0'. All other segments are blue and contain the digit '1'.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Reprezentarea binara a lui 190 in memoria calculatorului

$$190 = 2^7 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 \text{ (scrierea în baza 2 a lui 190)}$$

- ❑ cum se reprezintă -190 în memoria calculatorului?



A horizontal bar divided into 32 equal segments, each representing a bit of a 32-bit integer. The first segment is red and contains the digit '1'. All other segments are blue and contain the digit '0'.

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Reprezentarea binara a lui -190 in memoria calculatorului

- ❑ care este logica unei asemenea reprezentări?

Signed vs. unsigned

- ❑ **signed int** – întreg cu semn (pozitiv sau negativ)
 - ❑ **unsigned int** – întreg fără semn (pozitiv)



Signed vs. unsigned

- ❑ **signed int** – întreg cu semn (pozitiv sau negativ)
- ❑ **unsigned int** – întreg fără semn (pozitiv)

$$\begin{array}{r} 190 \\ + \\ -190 \\ \hline = \\ 0 \end{array}$$

Cuprinsul cursului de azi

1. Introducere în limbajul C. Structura unui program C
2. Tipuri de date fundamentale
3. Variabile și constante
4. Expresii și operatori

Variabile și constante

- stochează datele necesare programului
 - valorile stocate în memoria sistemului în mod transparent de către programator
 - referirea la aceste date se face prin numele lor simbolice, adică prin identificatori
- **variabilele** stochează date care pot fi modificate în timpul execuției
- **constantele** păstrează aceeași valoare (cea cu care au fost inițializate) până la terminarea programului

Variabile

- se characterizează printr-un nume (identifier), un tip, o valoare, adresa de memorie unde se află stocată valoarea variabilei, domeniu de vizibilitate
- oricărei variabile i se alocă un spațiu de memorie corespunzător tipului variabilei
- exemplu: `int notaExamen = 10;`
 - `int` = tipul variabilei (de obicei se va stoca pe 32 de biți)
 - `notaExamen` = numele variabilei
 - `10` = valoarea variabilei
 - `¬aExamen` = adresa din memorie unde se află stocată valoarea variabilei cu numele notaExamen

Domeniul de vizibilitate al variabilelor

- domeniul de vizibilitate al unei variabile = porțiunea de cod la carei execuție variabila respectivă este accesibilă
- variabile **locale** – vizibile local, numai în funcția sau blocul de instrucțiuni unde au fost declarate.
- variabile **globale** – vizibile global, din orice zonă a codului.
- parametri **formali** ai unei funcții se comportă asemenea unor variabile locale.

Variabile locale

- variabile definite în corpul unei funcții sau a unui bloc de instrucțiuni. Sunt locale (vizibile) acelei funcții sau bloc.

```
#include <stdio.h>
void f1()
{
    int x=10;
    printf("\nIn functia f1 valoarea lui x este %d \n",x);
}
void f2()
{
    int x=20;
    printf("In functia f2 valoarea lui x este %d \n",x);
}
int main()
{
    int x = 30;
    f1();
    f2();
    printf("In main valoarea lui x este %d \n \n",x);
    return 0;
}
```

```
In functia f1 valoarea lui x este 10
In functia f2 valoarea lui x este 20
In main valoarea lui x este 30
```

Variabile locale

- variabile definite în corpul unei funcții sau a unui bloc de instrucțiuni. Sunt locale (vizibile) acelei funcții sau bloc.

The screenshot shows a terminal window with the following content:

```
variabileLocale2.c      x
1 #include <stdio.h>
2
3 int main()
4 {
5     int i;
6     for(i=0;i<10;i++)
7     {
8         int j = 2*i;
9         printf("j = %d \n",j);
10    }
11
12    printf("j = %d \n",j);
13
14    return 0;
15 }
```

[Bogdan-Alexes-MacBook-Pro:curs2 bogdan\$ gcc variabileLocale2.c
variabileLocale2.c:12:24: error: use of undeclared identifier 'j'
 printf("j = %d \n",j);
 ^
1 error generated.

A red arrow points from the text "Eroare la linia 12:" to the line 12 of the code.

Eroare la linia 12: variabila j nu a fost declarată. Ea este vizibilă numai în blocul de instrucțiuni anterior.

Variabile locale

- variabile definite în corpul unei funcții sau a unui bloc de instrucțiuni. Sunt locale (vizibile) acelei funcții sau bloc.

```
variabileLocale2.c
1 #include <stdio.h>
2
3 int main()
4 {
5     int i;
6     for(i=0;i<10;i++)
7     {
8         int j = 2*i;
9         printf("j = %d \n",j);
10    }
11    int j = i;
12    printf("j = %d \n",j);
13
14    return 0;
15 }
```

j
= 0
= 2
= 4
= 6
= 8
= 10
= 12
= 14
= 16
= 18
= 10

Variabile globale

- ❑ se declară în afara oricărei funcții și sunt vizibile în întreg programul
- ❑ pot fi accesate de către orice zonă a codului
- ❑ orice expresie are acces la ele, indiferent de tipul blocului de cod în care se află expresia

Variabile globale

```
variabileGlobale.c  x
1 #include <stdio.h>
2
3 int x = 10;
4
5 void f1(int x)
6 {
7     x = x + 10;
8     printf("\nIn functia f1 valoarea lui x este %d \n",x);
9 }
10
11 void f2(int x)
12 {
13     x = x + 20;
14     printf("In functia f2 valoarea lui x este %d \n",x);
15 }
16
17 int main()
18 {
19     f1(x);
20     x = x*5;
21     f2(x);
22     printf("In main valoarea lui x este %d \n \n",x);
23     return 0;
24 }
```

Ce afișează programul?

In functia f1 valoarea lui x este 20
In functia f2 valoarea lui x este 70
In main valoarea lui x este 50

La apelul lui f1 și f2 se realizează o copie locală a lui x. După ieșirea din f1, copia se distrugе. Întrucât f1 nu întoarce nicio valoare, x rămâne cu aceeași valoare înainte de apelarea lui f1.

Variabile globale

```
variabileGlobale.c

1 #include <stdio.h>
2
3 int x = 10;
4
5 void f1(int x)
6 {
7     x = x + 10;
8     printf("\nIn functia f1 valoarea lui x este %d \n",x);
9 }
10
11 void f2(int x)
12 {
13     x = x + 20;
14     printf("In functia f2 valoarea lui x este %d \n",x);
15 }
16
17 int main()
18 {
19     f1(x);
20     int x = 5;
21     f2(x);
22     printf("In main valoarea lui x este %d \n \n",x);
23     return 0;
24 }
```

Ce afișează programul?

In functia f1 valoarea lui x este 20
In functia f2 valoarea lui x este 25
In main valoarea lui x este 5

Variabila locală ia locul variabilei globale

Constante întregi

- zecimale (baza 10; prima cifră nenulă): 1234
- octale (baza 8; prima cifră 0): 01234
- hexazecimale (baza 16, prefixul 0x sau 0X): 0xFA; 0XABBA
- efectul sufixului adăugat unei constante întregi (în funcție de valoare):
 - U sau u: unsigned int sau unsigned long int -> 52u, 400000U
 - L sau l: long int -> 52L, 32000L
 - UL sau uL sau Ul sau ul unsigned long int 52uL, 400000UL

Constante întregi

```
constanteIntregi.c

1 #include <stdio.h>
2
3 int main(){
4     int x;
5     x = 123;
6     printf("%d \n", x);
7
8     x = 0123;
9     printf("%d \n", x);
10
11    x = 0xAA;
12    printf("%d \n", x);
13
14    return 0;
15 }
```

Ce afișează programul?

123
83
170

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

Constante în virgulă mobilă

- ❑ compuse din semn, parte întreagă, punctul zecimal, parte fracționară, marcajul pentru exponent (e sau E)
- ❑ partea întreagă sau fracționară pot lipsi (dar nu ambele)
- ❑ punctul zecimal sau marcajul exponențial pot lipsi (dar nu ambele)
- ❑ **format aritmetic**: 3.1415
- ❑ **format exponențial**: 31415E-4,6.023E+23
- ❑ implicit constantele în virgulă mobilă sunt **stocate ca double**

Constante caracter

- ❑ au ca valoarea codul ASCII al caracterelor pe care le reprezintă
- ❑ caractere imprimabile: caractere grafice (coduri ASCII între 33 și 126) + spațiu (cod ASCII = 32)
- ❑ o constantă caracter corespunzătoare unui caracter imprimabil se reprezintă prin caracterul respectiv inclus între caractere apostrof: ‘a’ (codul ASCII 97), ‘A’ (codul ASCII 65)
- ❑ cum se reprezintă caracterul apostrof?
 - ❑ apostrof = ‘\’;
- ❑ cum se reprezintă caracterul backslash?
 - ❑ backslash = ‘\\’;

Constante caracter

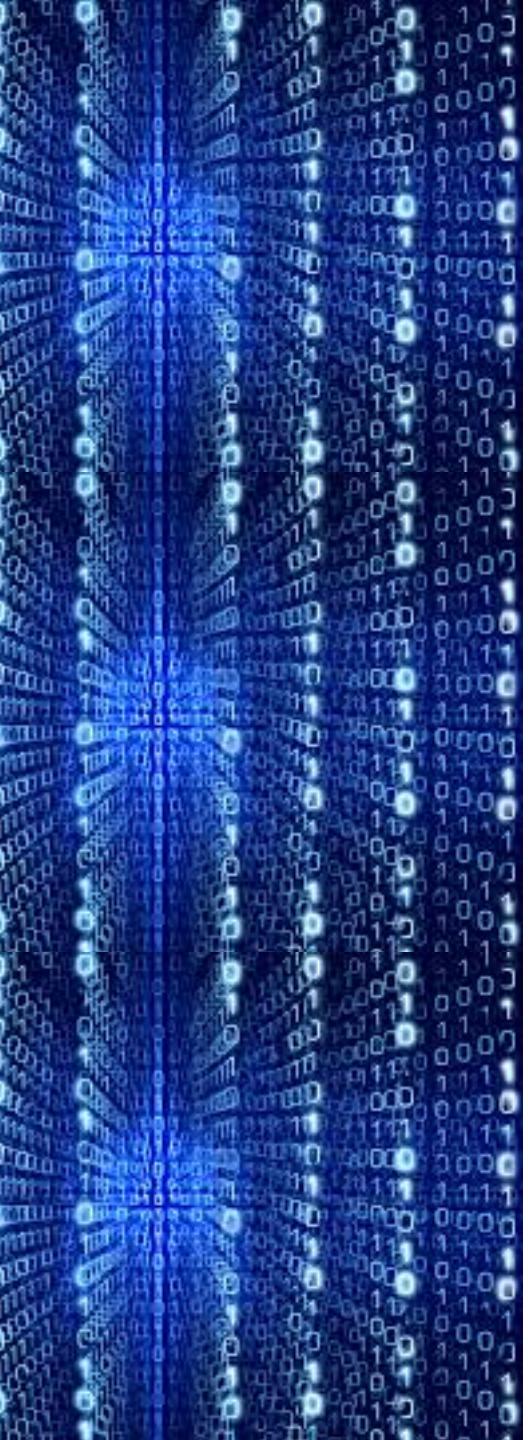
- au ca valoarea codul ASCII al caracterelor pe care le reprezintă
- caractere imprimabile: caractere grafice (coduri ASCII între 33 și 126) + spațiu (cod ASCII = 32)

- sevenete speciale (escape – de evitare a situațiilor care ar parea ambigue)

\a	BELL	generator de sunet
\b	BS	backspace
\f	FF	form feed
\n	LF	line feed
\r	CR	carriage return
\t	HT	Horizontal TAB
\v	VT	Vertical TAB
\\\	\	backslash
\'	'	apostrof
\"	"	ghilimele
\?	?	semnul ?
'\0'..'\0377'		orice caracter ASCII specificat OCTAL
'\0x0'..'\0xFF'		orice caracter ASCII specificat HEXAZECIMAL

Identifieri

- fiecare constantă și variabilă trebuie să aibă un nume unic
- reguli:
 - sunt permise doar literele alfabetului, cifrele și _(underscore)
 - identifierul nu poate începe cu o cifră
 - nu putem declara variabile având numele: 2win, etc.
 - literele mari sunt tratate diferit de literele mici
 - Maxim, maxim, maXim și MaxiM ar desemna variabile diferite
 - numele nu poate fi cuvânt cheie al limbajului C
 - nu putem declara variabile având numele **for**, **while**, **exit**, etc.



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,

2018-2019

Cursul 3

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, siruri 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)

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

□ Tablouri și pointeri

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

□ Siruri 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.
- Programare generică.

□ Recursivitate

Cuprinsul cursului de azi

1. Expresii și operatori
2. Conversii
3. Tipuri derivate de date: pointeri, tablouri, șiruri de caractere, structuri, uniuni, câmpuri de biți

Expresii și operatori

□ expresii

- sunt formate din **operanzi** și **operatori**;
- arată modul de calcul al unor valori;
- cea mai simplă expresie este formată dintr-un operand;

□ operatori

- elemente fundamentale ale expresiilor
- operatori aritmetici, relaționali, etc.
- C are foarte mulți operatori (46 în tabelul de la sfârșit)

□ operanzi

- variabilă, o constantă
- apel de funcție
- expresie între paranteze
- etc.

Expresii aritmetice și operatori aritmetici

Se aplică asupra unui singur operand

Unari		+ (plus unar) - (minus unar)
	<i>Aditivi</i>	+ (adunare) - (scădere)
Binari		* (înmulțire) / (împărțire) % (restul împărțirii)
	<i>Multiplicativi</i>	

Necesită doi operanzi

Expresii aritmetice și operatori aritmetici

□ exemple

```
int a, b, c = +3;          // operatorul unar +
b = -4;                   // operatorul unar -
a = b - c + 1;            // operatorul binar - și +      a este -6
a = a * b / 2;            // operatorul binar * și /      a este 12
c = a % 5;                // operatorul binar %          c este 2
```

- operatorii aritmetici se pot aplica asupra operanzilor
 - de tip **întreg** (int, char) sau
 - de tip **real** (float sau double)
- se pot **combina** aceste tipuri în aceeași expresie
 - **excepție**: % doar între întregi

Expresii aritmetice și operatori aritmetici

□ observații:

- operatorul / semnifică
 - împărțirea **întreagă** dacă ambii operanzi sunt întregi (int, char)
 - împărțirea **cu virgulă** dacă cel puțin unul dintre operanzi este de tip real (float, double)

```
int a = 5, b = 2;
float x = 5.0f;
a = a / b;      // a este 2
x = x / b;      // x este 2.5
x = 5 / b;      // x este 2.0
```

- împărțirea la zero !!
 - operatorii / și % nu pot avea operandul din dreapta 0
- trunchierea la împărțirea întreagă
 - C89 – dependent de implementare
 - C99 – trunchiere către 0

```
c = 7 / 5;          // c este 1 (trunchiat de la 1.4)
c = -7 / 5;         // c este -1 (trunchiat de la -1.4)
c = 9 / 5;          // c este 1 (trunchiat de la 1.8)
c = 9 / -5;         // c este -1 (trunchiat de la -1.8)
```

Evaluarea expresiilor

- introducem **principii fundamentale** pentru evaluarea oricăror expresii prin intermediul expresiilor aritmetice
 - mai ușor de înțeles astfel
- **precedență și asociativitatea** operatorilor
 - dacă într-o expresie apar mai mulți operatori, atunci evaluarea expresiei respectă **ordinea de precedență** a operatorilor
 - dacă într-o expresie apar mai mulți operatori de aceeași prioritate, atunci se aplică **regula de asociativitate** a operatorilor

Ordinea de precedență

□ ordinea de precedență a operatorilor aritmetici

Prioritate crescută	+ (plus unar) - (minus unar)
	* (înmulțire) / (împărțire) % (restul împărțirii)
Prioritate scăzută	+ (adunare) - (scădere)

□ exemple:

$$\begin{array}{l} a + b * c \\ -a * b - c \\ +a - b / c \end{array}$$

este echivalent cu
este echivalent cu
este echivalent cu

$$\begin{array}{l} a + (b * c) \\ ((-a) * b) - c \\ (+a) - (b / c) \end{array}$$

Regula de asociativitate

- regula de asociativitate a operatorilor aritmetici
 - un operator este **asociativ la stânga** dacă se grupează *de la stânga la dreapta*
 - exemple: toți operatorii aritmetici binari (+, -, *, /, %)

$a + b - c$	este echivalent cu	$(a + b) - c$
$a * b / c$	este echivalent cu	$(a * b) / c$

- un operator este **asociativ la dreapta** dacă se grupează *de la dreapta la stânga*
 - exemple: operatorii aritmetici unari (+, -)

$- + a$	este echivalent cu	$- (+a)$
$+ - a$	este echivalent cu	$+ (-a)$

Operatori

1. Operatori aritmetici
2. Operatorul de atribuire
3. Operatori de incrementare și decrementare
4. Operatori de egalitate, logici și relaționali
5. Operatori pe biți
6. Alți operatori:
 - de acces la elemente unui tablou, de apel de funcție, de adresa,
 - de referențiere, sizeof, de conversie explicită, condițional,
 - virgulă

Operatori de atribuire

□ operatorul de atribuire simplă =

- efect: evaluarea expresiei din dreapta operatorului și asignarea acestei valori la variabila din stânga operatorului

```
a = 10;           // a ia valoarea 10
b = a;            // b ia valoarea 10
c = a + (b-7) * 3; // c ia valoarea 19
```

□ valoarea unei atribuirii **var = expresie** este valoarea lui var după asignare

- expresia de atribuire poate apărea ca operand într-o altă expresie unde se așteaptă o valoare de tipul var

```
a = 3;
b = 5 - (c = a);      // c ia valoarea 3 care
                      // se scade din 5 și astfel b devine 2
```

- expresia devine greu de înțeles și poate introduce erori greu de depistat

Operatori de atribuire

- atribuirea formalizată: **expr1 = expr2**
 - expr1 este *lvalue* (valoare stânga)
 - trebuie să permită stocarea valorii lui expr2 în memorie
 - corect: $v[i+1] = 10$
 - incorrect: $10 = v[i+1]$
- dacă tipul lui **expr1** și **expr2** nu este același, atunci se aplică regula conversiei implice
 - valoarea lui **expr2** este convertită la tipul lui **expr1** în momentul asignării

```
int a;
float x;
a = 12.34f;           // a ia valoarea 12
x = 123;              // x ia valoarea 123.0
```

Operatori de atribuire

- ❑ regula de asociativitate
 - ❑ operatorul de atribuire este asociativ dreapta
 - ❑ atribuirile se pot înlántui

$$a = b = c = 0$$

- ❑ operatori de atribuire compuși (operator =)
 - ❑ exemplu : `+=`, `-+`, `*=`, `/=`, `%=`, și-md. (combinat cu operatori pe biți)
 - ❑ permit calcularea noii valori a variabilei folosind valoarea veche a acesteia

```
a += 1;           // a se incrementează cu 1: a = a + 1;  
b -= 3;           // asemănător cu b = b - 3;  
c *= 4;           // asemănător cu c = c * 4;
```

- dar nu este întotdeauna echivalent cu varianta descompusă
 - contează ordinea de precedență și efectele secundare

Operatori de incrementare și decrementare

- operatorii **++** și **--**
 - incrementarea/decrementarea unei variabile cu 1
 - specifici limbajelor de asamblare, mult mai rapizi
- forma **prefixă** (**++i** sau **--i**)
 - preincrementare/predecrementare
- forma **postfixă** (**i++** sau **i--**)
 - postincrementare/postdecrementare
- efect secundar: modificarea valorii operandului
- valoarea returnată
 - preincrementarea (**++a**) returnează valoarea **a+1**
 - postincrementarea (**a++**) returnează valoarea **a**

i++;

Exemplu echivalent:

i = i + 1;
i += 1;

Operatori de incrementare și decrementare

```
int a = 5, b = 2, c;  
c = a - ++b;           // ⇔ b = b+1;  c = a-b;  
                      // valorile a: 5, b: 3, c: 2  
c = ++a + b--;        // ⇔ a = a+1;  c = a + b;  b = b-1;  
                      // valorile a: 6, b: 2, c: 9
```

- operatorii de **preincrementare** și **predecrementare** au aceeași prioritate ca și operatorii unari + și - și sunt asociativi dreapta
- operatorii de **postincrementare** și **postdecrementare** au prioritate crescută în raport cu operatorii unari + și - și sunt asociativi stânga

Expresii logice

- ❑ expresiile logice se evaluatează la valori de tip *adevărat* sau *fals*
- ❑ sunt construite cu ajutorul a trei categorii de operatori
 - ❑ operatori **relaționali**
 - ❑ operatori de **egalitate**
 - ❑ operatori **logici**
- ❑ limbajul C tratează valorile *adevărat* și *fals* ca valori întregi
 - ❑ 0 înseamnă fals
 - ❑ orice altă valoare nenulă se interpretează ca adevărat

Operatori relaționali

- operatorii `<`, `>`, `<=`, `>=`
- rezultatul este o valoare logică, adică valoarea 0 (fals) sau 1 (adevărat)
- sunt mai puțin prioritari decât operatorii aritmetici și sunt asociativi stânga

```
5    < 10          // rezultat: 1
10   <  5          // rezultat: 0
3    > 2.5         // rezultat: 1
                  // se pot combina tipurile întreg și real
a + b <= c - 1  // este de fapt (a + b) <= (c - 1)
                  // respectând ordinea de precedență

a < b < c  // echivalent cu (a < b) < c
              // datorita asociativitatii stanga
```

Operatori de egalitate

- testează egalitatea dintre două valori
- **==** este operatorul "egal cu",
- **!=** este operatorul "diferit de"
- generează o valoare logică: 0 (fals) sau 1 (adevărat)
- sunt asociativi stânga
- în ordinea de precedență a operatorilor sunt mai puțin prioritari decât operatorii relaționali

```
a == 2           // returnează 1 dacă a este 2,  
                  // 0 în caz contrar  
a != b           // returnează 1 dacă a nu este egal cu b,  
                  // 0 dacă a și b au valori identice  
a < b == b < c  // este echivalent cu (a < b) == (b < c)  
                  // returnează 1 doar dacă expresiile au  
                  // aceeași valoare:  
                  // ambele sunt adevărate sau ambele false
```

Operatori logici

- limbajul C furnizează 3 operatori logici

- ! - operatorul unar, negare logică

```
!expr      // 1 dacă expr are valoarea logică 0 (fals)
           // 0 dacă expr are valoarea logică nenulă (adevărat)
```

- && - operator binar, **ȘI** logic

```
expr1 && expr2 // este 1 dacă expr1 și expr2 sunt nenule
```

- || - operator binar, **SAU** logic

```
expr1 || expr2 // este 1 dacă expr1 sau expr2 este nenulă
```

- generează o valoare logică: 0 (fals) sau 1 (adevărat)

Operatori logici

- evaluarea
 - dacă se poate deduce rezultatul global din evaluarea expresiei din stânga, atunci expresia din dreapta nu se mai evaluează

```
(a != 0) && (a % 4 == 0)
```

- operatorul ! (negare logică) are prioritate egală cu cea a operatorilor aritmetici unari (+ și -)
- operatorii && și || sunt mai puțin prioritari decât operatorii relaționali și cei de egalitate

Operatori pe biți

- două categorii
 - operatori **logici pe biți**
 - $\&$ **ȘI pe biți**, operator binar
 - $|$ **SAU pe biți**, operator binar
 - $^$ **SAU EXCLUSIV pe biți**, operator binar
 - \sim **complement față de 1**, operator unar
 - operatori de **deplasare pe biți**
 - $<<$ **deplasare stânga pe biți**, operator binar
 - $>>$ **deplasare dreapta pe biți**, operator binar
- operanzi de tip întreg (nu merg pe float, double)
- ordinea de precedență - în cadrul acestei categorii

Prioritate crescută	\sim (complement față de unu)
	$<<$ (deplasare stânga)
	$>>$ (deplasare dreapta)
	$\&$ (și pe biți)
	$^$ (sau exclusiv pe biți)
Prioritate scăzută	$ $ (sau pe biți)

Operatori pe biți

- $\&$ seamănă cu $\&\&$
- $|$ seamănă cu $||$
- au un rol similar, dar la nivelul fiecărei perechi de biți de pe poziții identice
- \sim este echivalentul operației $!$ dar aplicat la nivel de biți

Expresie	Reprezentare pe 4 biți				Observație
$a = 10$	1	0	1	0	
$b = 7$	0	1	1	1	
$a \& b$	0	0	1	0	1 dacă ambele biți sunt 1, 0 în rest
$a b$	1	1	1	1	1 dacă cel puțin unul din cei doi biți este 1, 0 în rest
$a ^ b$	1	1	0	1	1 dacă doar unul din cei doi biți este 1, 0 în rest
$\sim a$	0	1	0	1	1 unde bitul a fost 0 și 0 unde bitul a fost 1
$\sim b$	1	0	0	0	1 unde bitul a fost 0 și 0 unde bitul a fost 1

Operatori de deplasare pe biți

- condiții:
 - operanzi întregi
 - al doilea operand cu valoare mai mică (nu negativ) decât numărul de biți pe care este reprezentat operandul din stânga
- deplasarea spre stânga \Leftrightarrow înmulțire cu 2 la puterea deplasamentului
(în anumite condiții)
- deplasarea spre dreapta \Leftrightarrow împărțire cu 2 la puterea deplasamentului
(în anumite condiții)

Expresie	Reprezentare binară	Observație
a = 12	0000 0000 0000 1100	
b = 3600	0000 1110 0001 0000	
a << 1	0000 0000 0001 1000	Valoarea rezultată este $24 = 12 * 2^1$
a << 2	0000 0000 0011 0000	Valoarea rezultată este $48 = 12 * 2^2$
a << 5	0000 0001 1000 0000	Valoarea rezultată este $384 = 12 * 2^5$
a >> 1	0000 0000 0000 0110	Valoarea rezultată este $6 = 12 / 2^1$
a >> 2	0000 0000 0000 0011	Valoarea rezultată este $3 = 12 / 2^2$
b >> 4	0000 0000 1110 0001	Valoarea rezultată este $225 = 3600 / 2^4$

Operatori pe biți pe numere negative

Pagina 85 din standardul C99:

©ISO/IEC

ISO/IEC 9899:1999 (E)

- 4 The result of $\mathbf{E1} \ll \mathbf{E2}$ is $\mathbf{E1}$ left-shifted $\mathbf{E2}$ bit positions; vacated bits are filled with zeros. If $\mathbf{E1}$ has an unsigned type, the value of the result is $\mathbf{E1} \times 2^{\mathbf{E2}}$, reduced modulo one more than the maximum value representable in the result type. If $\mathbf{E1}$ has a signed type and nonnegative value, and $\mathbf{E1} \times 2^{\mathbf{E2}}$ is representable in the result type, then that is the resulting value; otherwise, the behavior is undefined.
- 5 The result of $\mathbf{E1} \gg \mathbf{E2}$ is $\mathbf{E1}$ right-shifted $\mathbf{E2}$ bit positions. If $\mathbf{E1}$ has an unsigned type or if $\mathbf{E1}$ has a signed type and a nonnegative value, the value of the result is the integral part of the quotient of $\mathbf{E1}$ divided by the quantity, 2 raised to the power $\mathbf{E2}$. If $\mathbf{E1}$ has a signed type and a negative value, the resulting value is implementation-defined.

Operatori pe biți pe numere negative

operatiiBitiNumereNegative1.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5
6 void afiseazaScriereBinara(int x)
7 {
8     //determină scrierea în baza 2
9     printf("Scrierea binara a lui %d este \n",x);
10    unsigned long long m = 1ULL << (8*sizeof(int)-1);
11    unsigned char b;
12    for(b = 0; b < 8*sizeof(int); b++){
13        printf("%u" , (x & m) != 0);
14        m = m >> 1;
15    }
16    printf("\n");
17 }
```

```
19 int main()
20 {
21
22     int a = -1;
23     afiseazaScriereBinara(a);
24
25     a = a<<1;
26     printf("%d \n",a);
27     afiseazaScriereBinara(a);
28
29     a = a<<2;
30     printf("%d \n",a);
31     afiseazaScriereBinara(a);
32
33     a = -1;
34     a = a>>1;
35     printf("%d \n",a);
36     afiseazaScriereBinara(a);
37
38     a = a>>2;
39     printf("%d \n",a);
40     afiseazaScriereBinara(a);
41
42
43     return 0;
44 }
```

Operatori pe biți pe numere negative

operatiiBitiNumereNegative1.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5
6 void afiseazaScriereBinara(int x)
7 {
8     //determina scrierea in baza 2
9     printf("Scrierea binara a lui %d este \n",x);
10    unsigned long long m = 1ULL << (8*sizeof(int)-1);
11    unsigned char b;
12    for(b = 0; b < 8*sizeof(int); b++){
13        printf("%u" , (x & m) != 0);
14        m = m >> 1;
15    }
16    printf("\n");
17 }
```

```
Scrierea binara a lui -1 este
11111111111111111111111111111111
-2
Scrierea binara a lui -2 este
11111111111111111111111111111110
-8
Scrierea binara a lui -8 este
111111111111111111111111111111000
-1
Scrierea binara a lui -1 este
11111111111111111111111111111111
-1
Scrierea binara a lui -1 este
11111111111111111111111111111111
-1
Scrierea binara a lui -1 este
11111111111111111111111111111111
```

```
19 int main()
20 {
21
22     int a = -1;
23     afiseazaScriereBinara(a);
24
25     a = a<<1;
26     printf("%d \n",a);
27     afiseazaScriereBinara(a);
28
29     a = a<<2;
30     printf("%d \n",a);
31     afiseazaScriereBinara(a);
32
33     a = -1;
34     a = a>>1;
35     printf("%d \n",a);
36     afiseazaScriereBinara(a);
37
38     a = a>>2;
39     printf("%d \n",a);
40     afiseazaScriereBinara(a);
41
42
43
44 }
```

Operatori pe biți pe numere negative

```
operatiiBitiNumereNegative2.c ●
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5
6 void afiseazaScriereBinara(int x)
7 {
8     //determina scrierea in baza 2
9     printf("Scrierea binara a lui %d este \n",x);
10    unsigned long long m = 1ULL << (8*sizeof(int)-1);
11    unsigned char b;
12    for(b = 0; b < 8*sizeof(int); b++){
13        printf("%u", (x & m) != 0);
14        m = m >> 1;
15    }
16    printf("\n");
17 }
18 }
```

```
19 int main()
20 {
21
22     int a = -(int) pow(2,31);
23     afiseazaScriereBinara(a);
24
25     a = a<<1;
26     printf("%d \n",a);
27     afiseazaScriereBinara(a);
28
29
30     a = -(int) pow(2,31);
31     a = a>>1;
32     printf("%d \n",a);
33     afiseazaScriereBinara(a);
34
35     a = a>>2;
36     printf("%d \n",a);
37     afiseazaScriereBinara(a);
38
39
40
41 }
```

Operatori pe biți pe numere negative

operatiiBitiNumereNegative2.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5
6 void afiseazaScriereBinara(int x)
7 {
8     //determina scrierea in baza 2
9     printf("Scrierea binara a lui %d este \n",x);
10    unsigned long long m = 1ULL << (8*sizeof(int)-1);
11    unsigned char b;
12    for(b = 0; b < 8*sizeof(int); b++){
13        printf("%u", (x & m) != 0);
14        m = m >> 1;
15    }
16    printf("\n");
17 }
```

```
Scrierea binara a lui -2147483648 este
10000000000000000000000000000000
0
Scrierea binara a lui 0 este
00000000000000000000000000000000
-1073741824
Scrierea binara a lui -1073741824 este
11000000000000000000000000000000
-268435456
Scrierea binara a lui -268435456 este
11110000000000000000000000000000
```

```
19 int main()
20 {
21
22     int a = -(int) pow(2,31);
23     afiseazaScriereBinara(a);
24
25     a = a<<1;
26     printf("%d \n",a);
27     afiseazaScriereBinara(a);
28
29     a = -(int) pow(2,31);
30     a = a>>1;
31     printf("%d \n",a);
32     afiseazaScriereBinara(a);
33
34     a = a>>2;
35     printf("%d \n",a);
36     afiseazaScriereBinara(a);
37
38
39     return 0;
40 }
```

Alți operatori

- ❑ operatorul de acces la elementele tabloului []

```
int a[100];  
a[5] = 10;
```
- ❑ operatorul de apel de funcție (): b = f(a);
- ❑ operatorul **adresă** & și operatorul de **dereferețiere** *
 - ❑ strâns legat de pointeri (cursurile următoare)

```
int a, *p;           // p este un pointer la int  
p = &a;             // p este pointer la a  
*p = 3;             // valoarea lui a devine 3
```

- ❑ operatorul **sizeof**

```
sizeof(a)           // este numărul de octeți  
// ocupări în memorie de a
```
- ❑ operatorul de conversie explicită: (tip)

```
int a = 1, b = 2;  
float media;  
  
media = ( a + (float)b ) / 2;    // media devine 1.5  
media = ( a + b ) / 2;          // media devine 1.0 - incorect!
```

Alți operatori

❑ operatorul condițional ? :

- ❑ operator ternar, decizional
- ❑ similar cu instrucțiunea **if**
- ❑ **expresie1? expresie2 : expresie3**
- ❑ dacă **expresie1** e adevarată, execută **expresie2**, altfel execută **expresie3**

```
int a=3, b=5, max;  
max = a > b ? a : b;  
  
a % 2 ? printf("numar impar") : printf("numar par");
```

❑ operatorul virgulă

- ❑ evaluarea secvențială a expresiilor (de la stânga la dreapta)
- ❑ valoarea ultimei expresii din înlănțuire este valoarea expresiei compuse
- ❑ cel mai puțin priorită din lista de precedență

```
int i, n, s;  
printf("n = ");scanf("%d",&n);  
for(i = 1,s = 0;i <= n;s = s + i,i = i + 1);
```

Ordinea de precedență și asociativitate

precedență	operatori	simbol	asociativitate
1	apel funcție / selecție	() [] . ->	SD
2	unari	* & - ! ~ ++ -- sizeof	DS
3	multiplicativi	* / %	SD
4	aditivi	+	SD
5	deplasări	<< >>	SD
6	relaționali	< > <= >=	SD
7	egalitate / neegalitate	= !=	SD
8	ȘI pe biți	&	SD
9	SAU exclusiv pe biți	^	SD
10	SAU inclusiv pe biți		SD
11	ȘI logic	&&	SD
12	SAU logic		SD
13	condițional	? :	DS
14	atribuire	= op=	DS
15	virgula	,	SD

Laboratorul 1 – rezolvări

2. Se citesc trei numere întregi de la tastatură. Să se afișeze maximul dintre cele 3 numere folosind operatorul decizional.
3. Se citește un număr întreg n de la tastatură. Să se calculeze $n*8$, $n/4$ și $n*10$ folosind operatorii logici de deplasare la nivel de bit.
4. Se citește un număr întreg de la tastatură. Să se determine dacă acesta este par sau impar folosind doar operatorii logici la nivel de biți.

```
solutiiLaborator1.c  x

1 #include <stdio.h>
2
3 int main()
4 {
5     //problema 2
6     int x,y,z;
7     scanf("%d%d%d",&x,&y,&z);
8     x > y ? x > z ? printf("x = %d e cel mai mare \n", x) : printf("z = %d e cel mai mare\n",z) :
9         y > z? printf("y = %d e cel mai mare \n", y) : printf("z = %d e cel mai mare\n",z);
10
11    //problema 3
12    int n;
13    scanf ("%d",&n);
14    printf("%d*8 = %d\n%d/4 = %d\n%d*10=%d\n",n,n<<3,n,n>>2,n, (n<<3) + (n<<1));
15
16    //problema 4
17    scanf("%d",&n);
18    n & 1 ? printf("%d e impar \n",n) : printf("%d e par \n",n);
19
20    //rezolvare gresita
21    n | 1 == n ? printf("%d e impar \n",n) : printf("%d e par \n",n);
22
23    return 0;
24 }
```

Laboratorul 1 – rezolvări

2. Se citesc trei numere întregi de la tastatură. Să se afișeze maximul dintre cele 3 numere folosind operatorul decizional.
3. Se citește un număr întreg n de la tastatură. Să se calculeze $n*8$, $n/4$ și $n*10$ folosind operatorii logici de deplasare la nivel de bit.
4. Se citește un număr întreg de la tastatură. Să se determine dacă acesta este par sau impar folosind doar operatorii logici la nivel de biți.

```
solutiiLaborator1.c  x
1 #include <stdio.h>
2
3 int main()
[Bogdan-Alexes-MacBook-Pro:diverse bogdan$ gcc solutiiLaborator1.c
solutiiLaborator1.c:21:4: warning: | has lower precedence than ==; == will be
      evaluated first [-Wparentheses]
        n | 1 == n ? printf("%d e impar \n",n) : printf("%d e par \n",n);
          ^~~~~~
solutiiLaborator1.c:21:4: note: place parentheses around the '==' expression to
      silence this warning
        n | 1 == n ? printf("%d e impar \n",n) : printf("%d e par \n",n);
          ^
16 //problema 4
17 scanf("%d",&n);
18 n & 1 ? printf("%d e impar \n",n) : printf("%d e par \n",n);
19
20 //rezolvare gresita
21 n | 1 == n ? printf("%d e impar \n",n) : printf("%d e par \n",n);
22
23 return 0;
24 }
```

Laboratorul 1 – rezolvări

```
solutiiLaborator1.c  x

1 #include <stdio.h>
2
3 int main()
4 {
5     //problema 2
6     int x,y,z;
7     scanf("%d%d%d",&x,&y,&z);
8     x > y ? x > z ? printf("x = %d e cel mai mare \n", x) : printf("z = %d e cel mai mare\n",z) :
9         y > z? printf("y = %d e cel mai mare \n", y) : printf("z = %d e cel mai mare\n",z);
10
11    //problema 3
12    int n;
13    scanf ("%d",&n);
14    printf("%d*8 = %d\n%d/4 = %d\n%d*10=%d\n",n,n<<3,n,n>>2,n, (n<<3) + (n<<1));
15
16    //problema 4
17    scanf("%d",&n);
18    n & 1 ? printf("%d e impar \n",n) : printf("%d e par \n",n);
19
20    //rezolvare gresita
21    n | 1 == n ? printf("%d e impar \n",n) : printf("%d e par \n",n);
22
23    //rezolvare corecta
24    (n | 1) == n ? printf("%d e impar \n",n) : printf("%d e par \n",n);
25
26    return 0;
27 }
```

Seminarul 2

- tema seminarului este operatori pe biți;
- are 9 pagini ☺;
- 7 probleme rezolvate (unele au și cod soluție);
- 9 probleme propuse studentilor (acestea se vor discuta la seminar);
- o să presupunem că ati citit problemele rezolvate ☺ (eventual puteti adresa intrebări legate de ele).

Cuprinsul cursului de azi

1. Expresii și operatori
2. Conversii
3. Tipuri derivate de date: pointeri, tablouri, siruri de caractere, structuri, uniuni, câmpuri de biți

Conversii implicate și explicite

```
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?

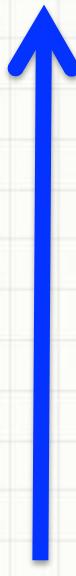
i = 4
i = 3
120
-1 > 1
-1 < 1

Conversii implicate

- 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 implicate - 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 doi 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 apărea 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

Cuprinsul cursului de azi

1. Expresii și operatori
2. Conversii
3. Tipuri derivate de date: **pointeri, tablouri, siruri de caractere, structuri, uniuni, câmpuri de biți**

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**

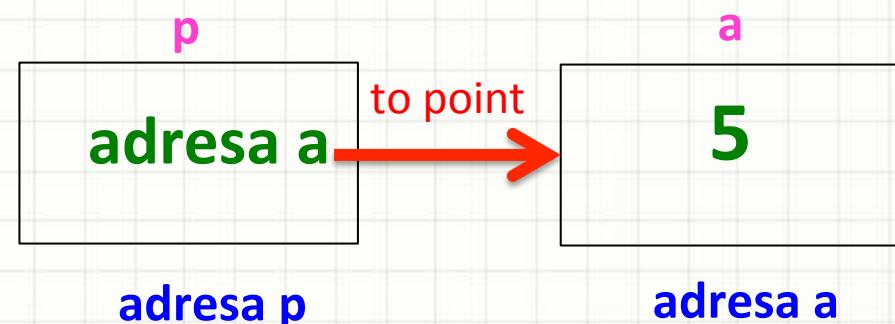
Pointeri

- 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;
```

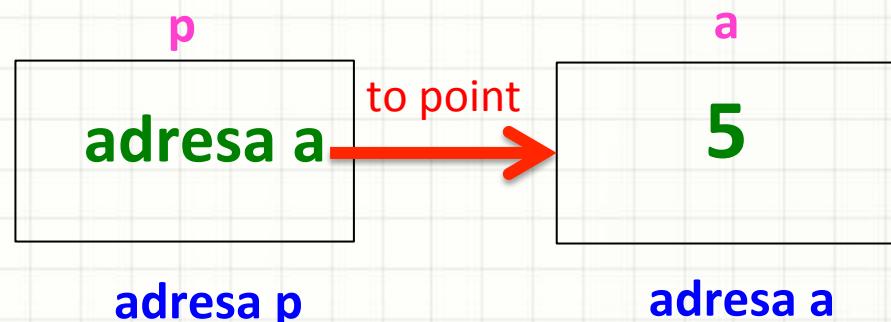


```
pointer1.c  
x  
1 #include <stdio.h>  
2  
3 int main() {  
4  
5     int a = 5, *p;  
6     p = &a;  
7     printf("Adresa lui a este: %p \n", &a);  
8     printf("Valoarea stocata la adresa lui a este: %d \n",a);  
9  
10    printf("Adresa lui p este: %p \n",&p);  
11    printf("Valoarea stocata la adresa lui p este: %p \n",p);  
12    printf("Valoarea variabilei a carei adresa e stocata in p este: %d \n",*p);  
13  
14    return 0;  
15 }
```

Pointeri

□ exemplu:

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



```
pointer1.c
```

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

```
Adresa lui a este: 0x7fff5a2eab58  
Valoarea stocata la adresa lui a este: 5  
Adresa lui p este: 0x7fff5a2eab50  
Valoarea stocata la adresa lui p este: 0x7fff5a2eab58  
Valoarea variabilei a carei adresa e stocata in p este: 5
```

Pointeri

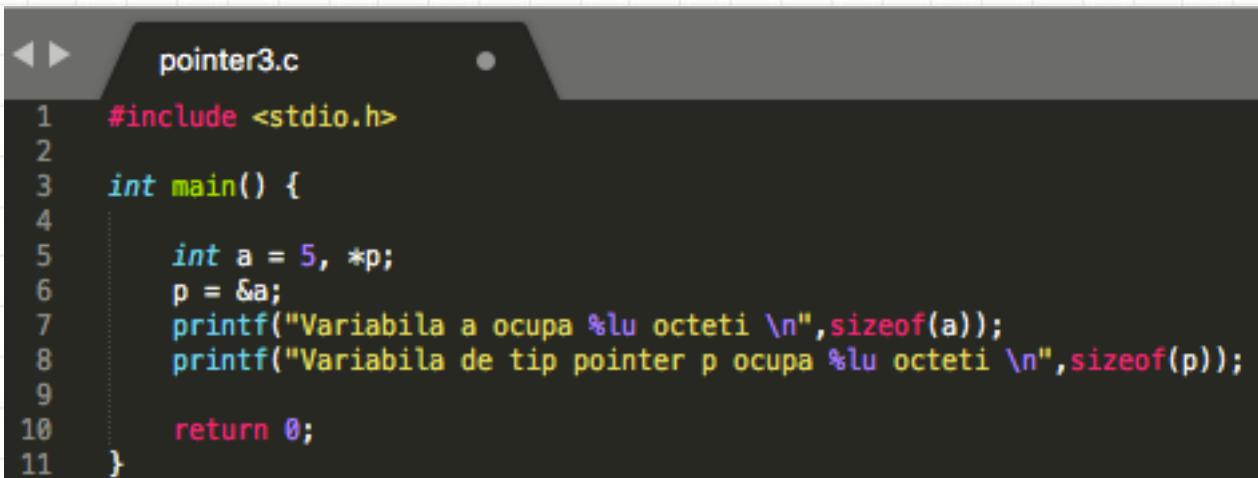
- exemplu: modificarea valorii unei variabile prin dereferențiere

```
pointer2.c
1 #include <stdio.h>
2
3 int main() {
4
5     int a = 5, *p;
6     p = &a;
7
8     printf("Valoarea stocata la adresa lui a este: %d \n",a);
9
10    a += 10; //acces direct
11    printf("Valoarea stocata la adresa lui a este: %d \n",a);
12
13    *p += 20; //acces indirect
14    printf("Valoarea stocata la adresa lui a este: %d \n",a);
15
16    return 0;
17 }
```

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

Pointeri

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



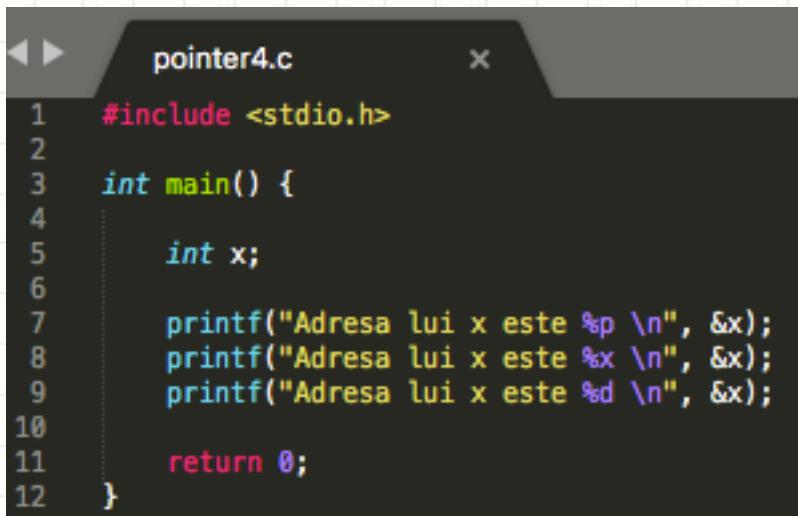
```
pointer3.c
1 #include <stdio.h>
2
3 int main() {
4
5     int a = 5, *p;
6     p = &a;
7     printf("Variabila a ocupa %lu octeti \n",sizeof(a));
8     printf("Variabila de tip pointer p ocupa %lu octeti \n",sizeof(p));
9
10    return 0;
11 }
```

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

Pointeri

□ adrese de memorie

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



```
#include <stdio.h>
int main() {
    int x;
    printf("Adresa lui x este %p \n", &x);
    printf("Adresa lui x este %x \n", &x);
    printf("Adresa lui x este %d \n", &x);
    return 0;
}
```

Adresa lui x este 0xffff5cca4b58
Adresa lui x este 5cca4b58
Adresa lui x este 1556761432

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 X". The address bar contains the URL "www.unitconversion.org/numbers/base-10-to-base-16-conversion.html". Below the address bar, there are two input fields. The first input field is labeled "base-10:" and contains the value "1556761432". The second input field is labeled "base-16:" and contains the value "5CCA4B58".

Adresa lui x este 0xffff5cca4b58
Adresa lui x este 5cca4b58
Adresa lui x este 1556761432

Cuprinsul cursului de azi

1. Expresii și operatori
2. Conversii
3. Tipuri derivate de date: pointeri, **tablouri**, siruri de caractere, structuri, uniuni, câmpuri de biți

Tablouri unidimensionale

- **sintaxă:**

tip nume_tablou [dimensiune];

- **exemple:**

double v[100];

0.3	-1.2	10	5.7	...	0.2	-1.5	1
-----	------	----	-----	-----	-----	------	---

v[3] = 5.7;

0 1 2 3 97 98 99

int a[5];

3	-12	10	7	1
---	-----	----	---	---

a[0] = 3;

0 1 2 3 4

char c[34];

A	&	*	+	...	c	M	#
---	---	---	---	-----	---	---	---

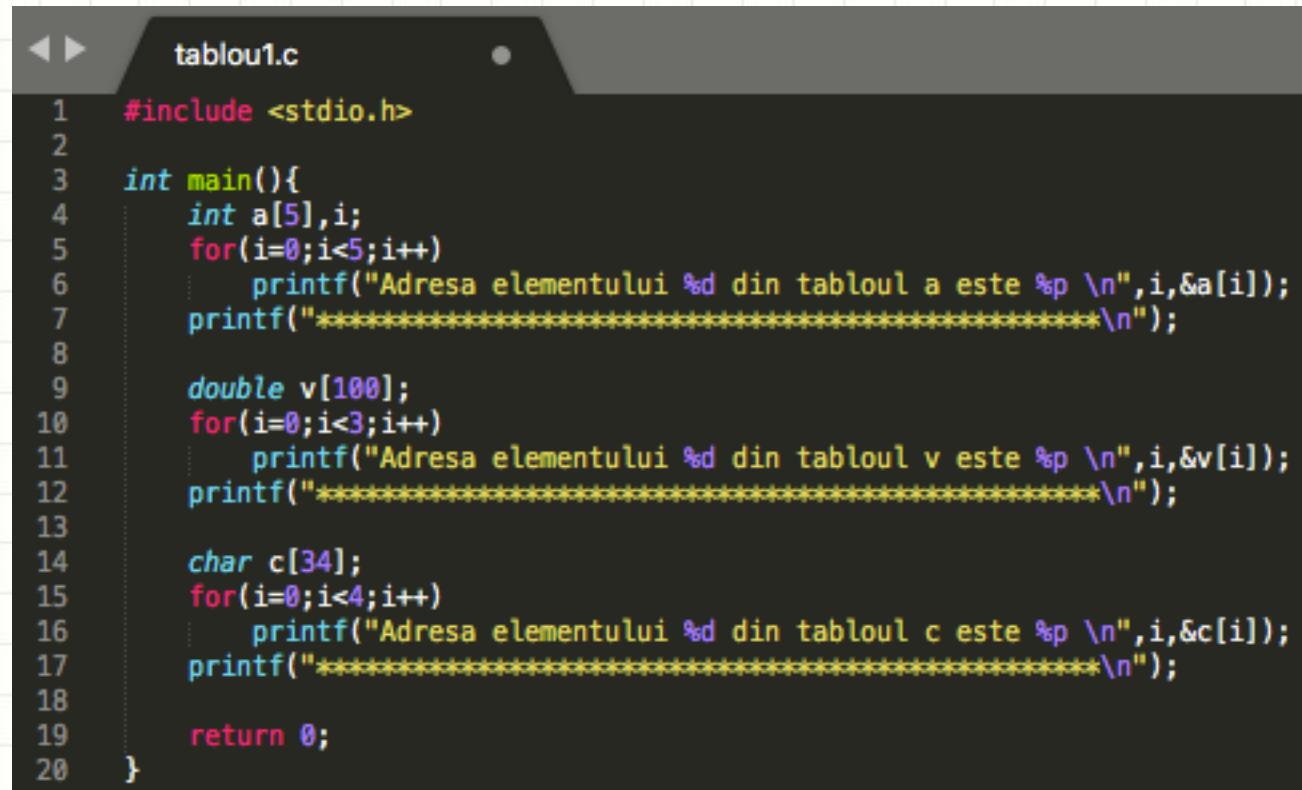
c[1] = '&';

0 1 2 3 31 32 33

- 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 }
```

Tablouri unidimensionale

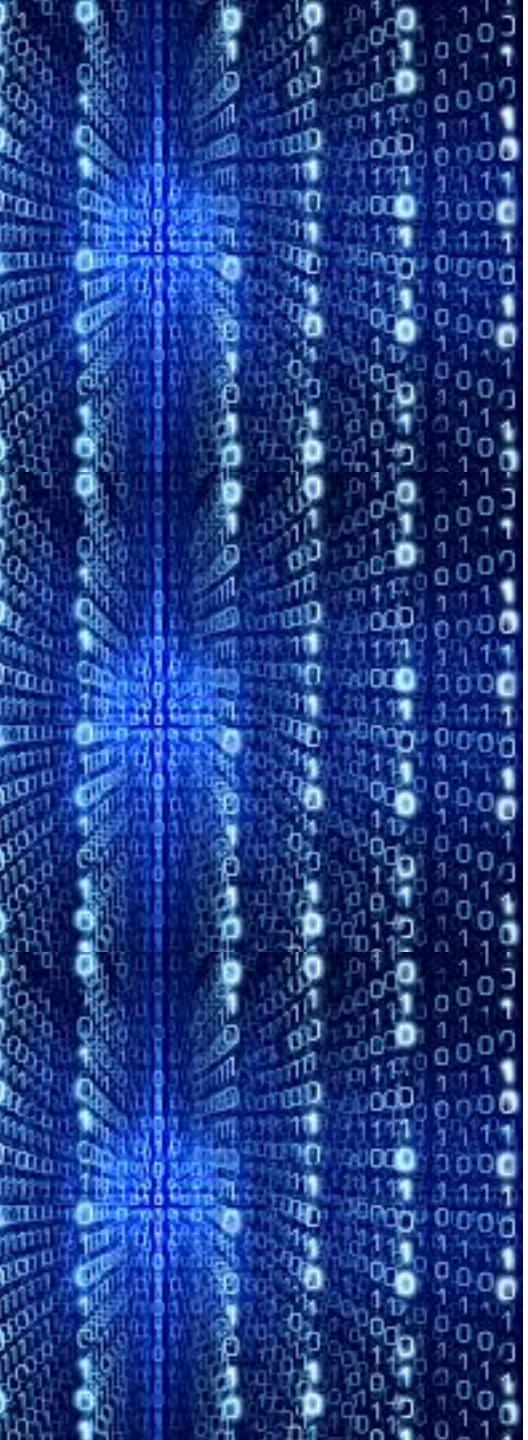
- definiție: set de valori de același tip memorat la adrese succesive de memorie.

```
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 }
```

Adresa elementului 0 din tabloul a este 0x7fff52d5bb40
Adresa elementului 1 din tabloul a este 0x7fff52d5bb44
Adresa elementului 2 din tabloul a este 0x7fff52d5bb48
Adresa elementului 3 din tabloul a este 0x7fff52d5bb4c
Adresa elementului 4 din tabloul a este 0x7fff52d5bb50

Adresa elementului 0 din tabloul v este 0x7fff52d5b820
Adresa elementului 1 din tabloul v este 0x7fff52d5b828
Adresa elementului 2 din tabloul v este 0x7fff52d5b830

Adresa elementului 0 din tabloul c este 0x7fff52d5b7f0
Adresa elementului 1 din tabloul c este 0x7fff52d5b7f1
Adresa elementului 2 din tabloul c este 0x7fff52d5b7f2
Adresa elementului 3 din tabloul c este 0x7fff52d5b7f3



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,

2018-2019

Cursul 4

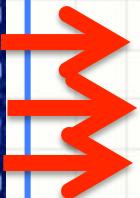
Programa cursului

□ 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, siruri 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)

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

□ Tablouri și pointeri

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

□ Siruri 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.
- Programare generică.

□ Recursivitate

Cuprinsul cursului de azi

1. Tipuri derivate de date: tablouri, siruri de caractere, structuri, campuri de biti, uniuni, enumerari, tipuri definite de utilizator (typedef).
2. Instructiuni de control.
3. Directive de procesare. Macrodefinitii.

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 a 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 a este %p \n",i,&c[i]);
17     printf("*****\n");
18
19     return 0;
20 }
```

Adresa elementului 0 din tabloul a este 0x7fff52d5bb40
Adresa elementului 1 din tabloul a este 0x7fff52d5bb44
Adresa elementului 2 din tabloul a este 0x7fff52d5bb48
Adresa elementului 3 din tabloul a este 0x7fff52d5bb4c
Adresa elementului 4 din tabloul a este 0x7fff52d5bb50

Adresa elementului 0 din tabloul v este 0x7fff52d5b820
Adresa elementului 1 din tabloul v este 0x7fff52d5b828
Adresa elementului 2 din tabloul v este 0x7fff52d5b830

Adresa elementului 0 din tabloul c este 0x7fff52d5b7f0
Adresa elementului 1 din tabloul c este 0x7fff52d5b7f1
Adresa elementului 2 din tabloul c este 0x7fff52d5b7f2
Adresa elementului 3 din tabloul c este 0x7fff52d5b7f3

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 ;**

```
tablou2.c
1 #include <stdio.h>
2
3 int main(){
4
5     double v[100];
6     int a[5];
7     char c[34];
8
9     printf("Stocarea tabloului v necesita %lu octeti \n",sizeof(v));
10    printf("Stocarea tabloului a necesita %lu octeti \n",sizeof(a));
11    printf("Stocarea tabloului c necesita %lu octeti \n",sizeof(c));
12
13    return 0;
14 }
```

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 ;**

The screenshot shows a terminal window with the file "tablou2.c" open. The code defines three arrays: v (double, 100 elements), a (int, 5 elements), and c (char, 34 elements). It then prints the size of each array in bytes using the `sizeof` operator. The output shows the sizes for each array.

```
tablou2.c
1 #include <stdio.h>
2
3 int main(){
4
5     double v[100];
6     int a[5];
7     char c[34];
8
9     printf("Stocarea tabloului v necesita %lu octeti \n",sizeof(v));
10    printf("Stocarea tabloului a necesita %lu octeti \n",sizeof(a));
11    printf("Stocarea tabloului c necesita %lu 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

Tablouri unidimensionale

□ exemplu de initializare

```
initializare.c

1 #include <stdio.h>
2
3 int main()
4 {
5     int nrElemente, i;
6     int a1[] = {1,2,3};
7     printf("Tabloul a1 are %d elemente\n",nrElemente = sizeof(a1)/sizeof(int));
8     for(i=0;i<nrElemente;i++)
9         printf("a1[%d] = %d\t",i,a1[i]);
10
11                                         Tabloul a1 are 3 elemente
12                                         a1[0] = 1           a1[1] = 2           a1[2] = 3
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]);
18
19                                         Tabloul a2 are 6 elemente
20                                         a2[0] = 1           a2[1] = 2           a2[2] = 3           a2[3] = 0           a2[4] = 0           a2[5] = 0
21
22     int a3[6] = {0};
23     printf("Tabloul a3 are %d elemente\n",nrElemente = sizeof(a3)/sizeof(int));
24     for(i=0;i<nrElemente;i++)
25         printf("a3[%d] = %d\t",i,a3[i]);
26
27                                         Tabloul a3 are 6 elemente
28                                         a3[0] = 0           a3[1] = 0           a3[2] = 0           a3[3] = 0           a3[4] = 0           a3[5] = 0
```

Tablouri unidimensionale

□ exemplu de initializare

```
30
31  int a4[6] = {10};
32  printf("Tabloul a4 are %d elemente\n",nrElemente = sizeof(a4)/sizeof(int));
33  for(i=0;i<nrElemente;i++)
34  printf("a4[%d] = %d\t",i,a4[i]);
35
Tabloul a4 are 6 elemente
a4[0] = 10      a4[1] = 0      a4[2] = 0      a4[3] = 0      a4[4] = 0      a4[5] = 0
-----
```



```
36
40  int a5[6] = {1,[2]=4,[3]=7,9};
41  printf("Tabloul a5 are %d elemente\n",nrElemente = sizeof(a5)/sizeof(int));
42  for(i=0;i<nrElemente;i++)
43  printf("a5[%d] = %d\t",i,a5[i]);
44
Tabloul a5 are 6 elemente
a5[0] = 1      a5[1] = 0      a5[2] = 4      a5[3] = 7      a5[4] = 9      a5[5] = 0
-----
```



```
45
46  int a6[6] = {1,[2] = 4, [1] = 7,9};
47  printf("Tabloul a6 are %d elemente\n",nrElemente = sizeof(a6)/sizeof(int));
48  for(i=0;i<nrElemente;i++)
49  printf("a6[%d] = %d\t",i,a6[i]);
50
Tabloul a6 are 6 elemente
a6[0] = 1      a6[1] = 7      a6[2] = 9      a6[3] = 0      a6[4] = 0      a6[5] = 0
-----
```

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

- **sintaxa**

tip nume_variabila [dimensiune1][dimensiune2];

- **tablou bidimensional = tablou de tablouri**

```
tablou2d.c

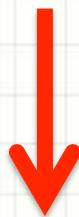
1 #include <stdio.h>
2
3 int main()
4 {
5
6     int a[3][5];
7
8     printf("%lu \n", sizeof(a[1][2]));
9     printf("%lu \n", sizeof(a));
10    printf("%lu \n", sizeof(a[0])));
11
12    return 0;
13 }
```

```
4
60
20
```

Tablouri bidimensionale

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

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

Cuprinsul cursului de azi

1. Tipuri derivate de date: tablouri, siruri de caractere, structuri, campuri de biti, uniuni, enumerari, tipuri definite de utilizator (typedef).
2. Instructiuni de control.
3. Directive de procesare. Macrodefinitii.

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 (reține 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)

```
afiseazaCaracter.c
1 #include <stdio.h>
2
3 int main()
4 {
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
15     printf("\n");
16     return 0;
17 }
```

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
{		}	~						

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	₣	214	₲	230	μ	246	÷
135	ç	151	ù	167	º	183		199		215	₩	231	τ	247	≈
136	è	152	ÿ	168	¸	184	_	200	Ł	216	+	232	Φ	248	°
137	ë	153	Ö	169	Ŕ	185		201	₹	217	Ј	233	ଓ	249	.
138	è	154	Ü	170	Ŕ	186		202	₩₩	218	₲	234	₪	250	.
139	ï	155	¢	171	½	187	_	203	₩	219	■	235	฿	251	√
140	î	156	£	172	¼	188	_	204	₣	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	■

Tipuri de date structurate

Limbajul C permite creare tipurilor de date în 5 moduri:

- structura (**struct**) – grupează mai multe variabile sub același nume, creează un nou tip de date agregând tipuri diferite de date;
- câmpul de biți (variațiune a structurii) → acces ușor la bitii individuali;
- uniunea (**union**) – face posibil ca aceleasi 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.

Cuprinsul cursului de azi

1. Tipuri derivate de date: tablouri, siruri de caractere, structuri, campuri de biti, uniuni, enumerari, tipuri definite de utilizator (typedef).
2. Instructiuni de control.
3. Directive de procesare. Macrodefinitii.

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 se face cu operatorul de selecție punct **.** care precizează identificatorul variabilei și al câmpului.

Structuri

□ exemplu:

```
struct student {  
    char nume[30];  
    char prenume[30];  
    float notaExamen;  
} A, B, C;
```

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

```
struct {  
    char nume[30];  
    char prenume[30];  
    float notaExamen;  
} A;
```

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

```
typedef struct {  
    char nume[30];  
    char prenume[30];  
    float notaExamen;  
} student;  
student A;
```

*Definește un tip de date numit **student** și declară ca fiind de acest tip variabila **A***

Structuri – inițializare

□ exemplu:

```
student.c x
1 #include<stdio.h>
2
3
4 typedef struct {
5     char nume[30];
6     char prenume[30];
7     float notaExamen;
8 } student;
9
10 int main()
11 {
12     student A = {"Popescu","Maria",9.25};
13     student B = {.notaExamen = 9.25,.prenume = "Maria",.nume = "Popescu"};
14     student C = {"Popescu"};
15
16     printf("%s %s %f \n", A.nume,A.prenume,A.notaExamen);
17     printf("%s %s %f \n", B.nume,B.prenume,B.notaExamen);
18     printf("%s %s %f \n", C.nume,C.prenume,C.notaExamen);
19
20     return 0;
21 }
```

```
typedef struct {
    char nume[30];
    char prenume[30];
    float notaExamen;
} student;
```

```
[Bogdan-Alexes-MacBook-Pro:curs4 bogdan$ gcc student.c
[Bogdan-Alexes-MacBook-Pro:curs4 bogdan$ ./a.out
Popescu Maria 9.250000
Popescu Maria 9.250000
Popescu 0.000000
```

Structuri imbricate și tablouri de structuri

- ❑ o structură este imbricată (nested) dacă ea conține ca membru o altă structură

```
struct student{  
    char nume[30];  
    char prenume[30];  
    float notaExamen;  
    struct adresa;  
}
```

Structura **adresa** trebuie să fie definită în prealabil

- ❑ tablou de structuri: tablou cu elemente de tip struct
struct student grupa[30];

Structuri

Operații permise cu structuri:

- **accesul la membri structurii** se face prin folosirea operatorului punct:
`nume_variabila.nume_camp`
- **atribuiri** pentru variabile de tip structură: `B = A;`
- **obținerea adresei** unei variabile structură
- **utilizarea operatorului sizeof** pentru determinarea dimensiunii unei structuri

Structuri

- definirea unei structuri nu ocupă memorie ci doar creează un tip nou de date
 - variabile declarate de tipul structurii respective ocupă memorie
 - dimensiunea memoriei ocupată de o astfel de variabilă este aproximativ suma dimensiunilor de memorie ocupată de fiecare componentă
 - zona de memorie ocupată este în final aliniată (se introduc, dacă este necesar, octeți în plus – padding bytes) pentru a facilita accesul la date (citire)

Structuri

```
aliniereStructuri.c  x

1 #include <stdio.h>
2
3 // char          1 byte
4 // short int    2 bytes
5 // int           4 bytes
6 // double        8 bytes
7
8 struct
9 {
10     char          c;
11     short int    s;
12 } A;
```

```
14     struct
15     {
16         short int    s;
17         char          c;
18         int           i;
19     } B;
20
21     struct
22     {
23         char          c;
24         double        d;
25         int           s;
26     } C;
27
28     struct
29     {
30         double        d;
31         int           s;
32         char          c;
33     } D;
```

```
35     int main()
36     {
37         printf("sizeof(A) = %d\n", sizeof(A));
38         printf("sizeof(B) = %d\n", sizeof(B));
39         printf("sizeof(C) = %d\n", sizeof(C));
40         printf("sizeof(D) = %d\n", sizeof(D));
41
42         return 0;
43     }
```

```
Program: aliniereStructuri.c
sizeof(A) = 4
sizeof(B) = 8
sizeof(C) = 24
sizeof(D) = 16
```

Cuprinsul cursului de azi

1. Tipuri derivate de date: tablouri, siruri de caractere, structuri, **câmpuri de biți**, uniuni, enumerări, tipuri definite de utilizator (typedef).
2. Instructiuni de control.
3. Directive de procesare. Macrodefiniții.

Câmpuri de biți

- tip special de membru al unei structuri care definește cât de lung trebuie să fie câmpul, în biți.
- putem stoca mai multe variabile într-un singur octet.
- nu se poate obține adresa unui câmp de biți.
- **ajută la economisirea memoriei utilizate.**

□ 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:

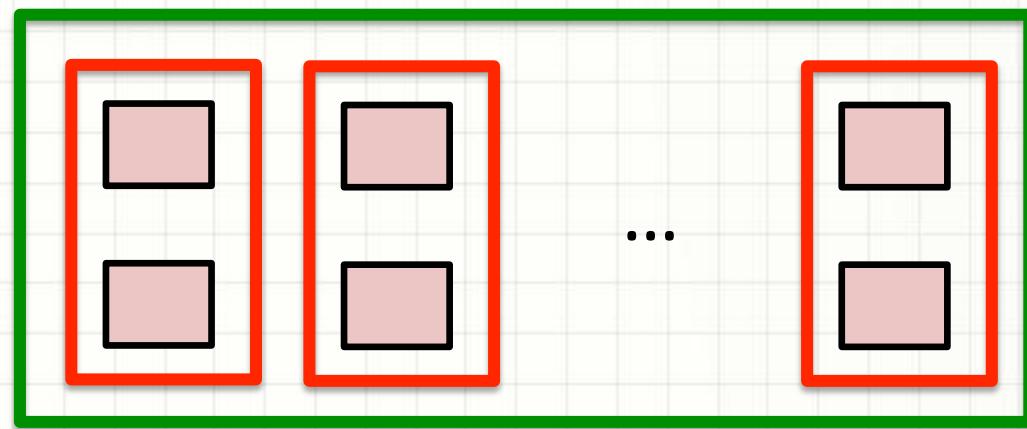
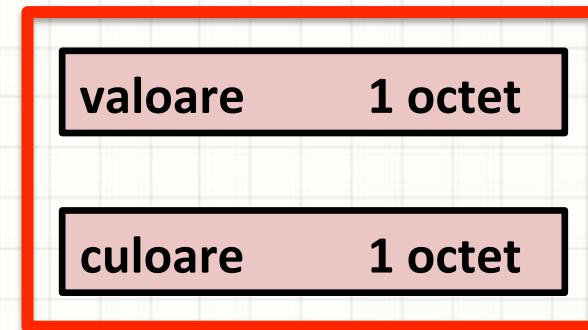
- tipul câmpului de biți poate fi doar întreg: **char**, **int**, **unsigned** sau **signed**.
- câmpul de biți cu lungimea 1 → **unsigned** (un singur bit nu poate avea semn).
- unele compilatoare → doar **unsigned**.
- lungime → numărul de biți dintr-un câmp.

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  
} A;
```

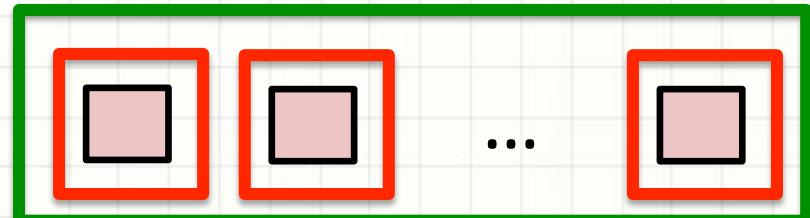
```
struct carteJoc PachetCartiJoc[52];
```



dimensiune A este 2
dimensiune pachetCartiJoc este 104

Câmpuri de biți

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



dimensiune A este 1

dimensiune pachetCartiJoc este 52

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.

Cuprinsul cursului de azi

1. Tipuri derivate de date: tablouri, siruri de caractere, structuri, campuri de biti, uniuni, enumerari, tipuri definite de utilizator (typedef).
2. Instructiuni de control.
3. Directive de procesare. Macrodefinitii.

Uniuni

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

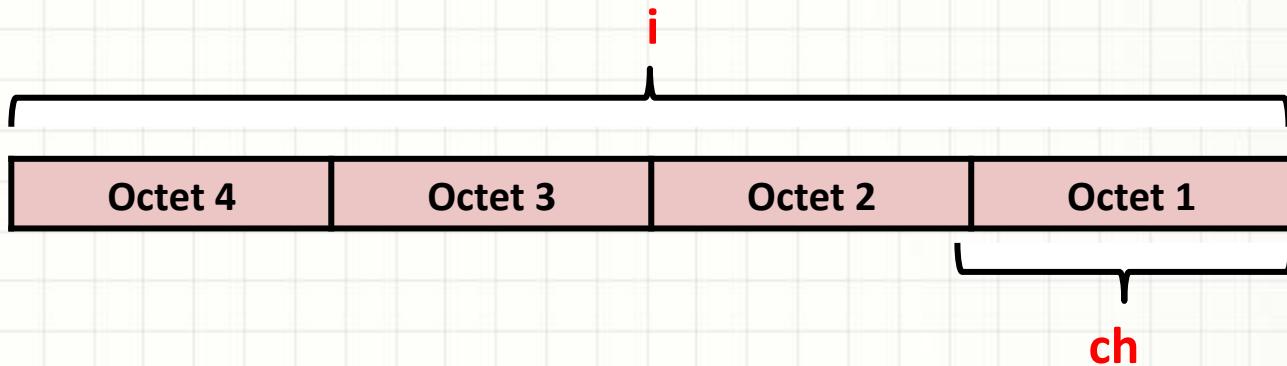
Sintaxa:

```
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

Octet 4

Octet 3

Octet 2

Octet 1

i

ch

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

Dimensiunea lui A este 4
A.ch = 1
A.i = 1
A.ch = 44
A.i = 300

Cuprinsul cursului de azi

1. Tipuri derivate de date: tablouri, siruri de caractere, structuri, campuri de biti, uniuni, enumerari, tipuri definite de utilizator (typedef).
2. Instructiuni de control.
3. Directive de procesare. Macrodefinitii.

Enumerări

- ❑ multime de constante de tip întreg care specifică toate valorile permise pe care le poate avea o variabilă de acel tip
- ❑ atât numele generic al enumerării cât și lista de variabile sunt optionale
- ❑ constanta unui element al enumerării este fie asociată implicit, fie explicit. Implicit, primul element are asociată valoarea 0, iar pentru restul este valoarea precedentă+1.

❑ **sintaxa:**

```
enum <nume> {  
    lista enumerarilor  
} lista variabile;
```

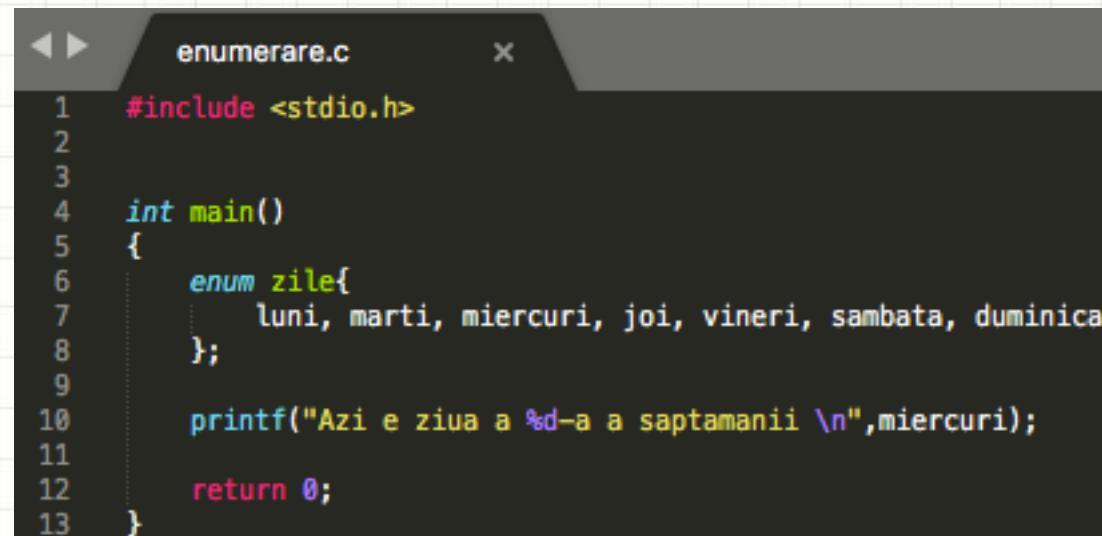
Enumerări

□ exemplu

enum {a, b, c, d}; → a = 0, b = 1, c = 2, d = 3

enum {a, b, c=7, d}; → a = 0, b = 1, c = 7, d = 8

enum {a=4, b=-3, c=9, d=-8}; enum {false,true};



```
enumerare.c
1 #include <stdio.h>
2
3
4 int main()
5 {
6     enum zile{
7         luni, marti, miercuri, joi, vineri, sambata, duminica
8     };
9
10    printf("Azi e ziua a %d-a a saptamanii \n",miercuri);
11
12    return 0;
13 }
```

[Bogdan-Alexes-MacBook-Pro:curs4 bogdan\$./a.out
Azi e ziua a 2-a a saptamanii

Enumerări

□ exemplu

enum {a, b, c, d}; → a = 0, b = 1, c = 2, d = 3

enum {a, b, c=7, d}; → a = 0, b = 1, c = 7, d = 8

enum {a=4, b=-3, c=9, d=-8}; enum {false,true};

The screenshot shows a terminal window with the following content:

```
enumerare.c
1 #include <stdio.h>
2
3
4 int main()
5 {
6     enum zile{
7         luni = 1, marti, miercuri, joi, vineri, sambata, duminica
8     };
9
10    printf("Azi e ziua a %d-a a saptamanii \n",miercuri);
11
12    return 0;
13 }
```

Bogdan-Alexes-MacBook-Pro:curs4 bogdan\$./a.out
Azi e ziua a 3-a a saptamanii

Cuprinsul cursului de azi

1. Tipuri derivate de date: tablouri, siruri de caractere, structuri, campuri de biti, uniuni, enumerari, tipuri definite de utilizator (typedef).
2. Instructiuni de control.
3. Directive de procesare. Macrodefinitii.

Specificatorul `typedef`

- definește explicit noi tipuri de date.
- nu se declară o variabilă sau o funcție de un anumit tip, ci se asociază un nume (sinonimul) tipului de date.
- **sintaxa:**

`typedef <definiție tip> <identificator>;`

- **exemple:**

`typedef unsigned int natural;`

`typedef long double tablouNumereReale [100] ;`

`tablouNumereReale a, b, c;`

`natural m,n,i;`

```
typedef struct {  
    char nume[30];  
    char prenume[30];  
    float notaExamen;  
} student;  
student A;
```

Subiectul de examen din septembrie 2018

Considerăm un pachet de cărți de joc uzual care conține 52 de cărți. Fiecare carte are o valoare din mulțimea ordonată $\{2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A\}$ și o culoare din mulțimea $\{h, d, s, c\}$ (hearts – inimă roșie, diamonds – romb, spades – inimă neagră, clubs – treflă).

Clasificarea mâinilor de poker. În jocul de poker, mâinile (mulțime de 5 cărți de joc diferite între ele) sunt clasificate într-un mod clar. În total sunt 10 clase diferite de mâini de poker care pot exista. Acestea pornesc de la “*chinta roială*”, care este cea mai valoroasa combinație posibilă, și descresc până la “*carte mare*”. Clasificarea este exemplificată în cele ce urmează pe baza unei figuri conținând exemple particulare din cele 10 clase posibile de mâini de poker ierarhizate.

10 h	J h	Q h	K h	A h
5 s	6 s	7 s	8 s	9 s
A h	A c	A d	A s	2 h
A s	A d	A c	K h	K s
2 h	4 h	5 h	8 h	K h

1. *Chintă roială* – conține cărțile cu valorile 10 (decar), J (valet), Q (damă), K (popă), A (as) de aceeași culoare.
2. *Chintă de culoare* – conține cinci cărți de valori consecutive, toate de aceeași culoare.
3. *Careu* – conține patru cărți de aceeași valoare și o carte adițională.
4. *Full* – conține trei cărți de aceeași valoare și alte două cărți având o altă valoare, dar egale între ele.
5. *Culoare* – conține cinci cărți de aceeași culoare.

Subiectul de examen din septembrie 2018

Considerăm un pachet de cărți de joc uzual care conține 52 de cărți. Fiecare carte are o valoare din mulțimea ordonată $\{2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A\}$ și o culoare din mulțimea $\{h, d, s, c\}$ (hearts – inimă roșie, diamonds – romb, spades – inimă neagră, clubs – treflă).

Clasificarea mâinilor de poker. În jocul de poker, mâinile (mulțime de 5 cărți de joc diferite între ele) sunt clasificate într-un mod clar. În total sunt 10 clase diferite de mâini de poker care pot exista. Acestea pornesc de la “*chinta roială*”, care este cea mai valoroasa combinație posibilă, și descresc până la “*carte mare*”. Clasificarea este exemplificată în cele ce urmează pe baza unei figuri conținând exemple particulare din cele 10 clase posibile de mâini de poker ierarhizate.



6. *Chintă* – conține cinci cărți de valori consecutive.

7. *Trei de un fel* – conține trei cărți având aceeași valoare, alături de alte două cărți de valori diferite.

8. *Două perechi* – conține două cărți având aceeași valoare, alte două cărți având aceeași valoare (dar diferită de valoarea primelor două cărți) și o a cincea carte de valoare diferită.

9. *O pereche* – conține două cărți având aceeași valoare, alături de alte trei cărți de valori diferite.

10. *Carte mare* - orice mână care nu intră în niciuna dintre clasele de mai sus.

Subiectul de examen din septembrie 2018

Reprezentarea mâinilor în fișiere text. Fiecare mână este codificată într-un fișier text printr-o linie conținând cinci cărți separate printr-un singur spațiu. Fiecare carte este reprezentată de valoarea sa (numerele de la 2 la 10 sau caracterele J, Q, K, A) și de culoarea sa. Spre exemplu, mâinile din figura anterioară se reprezintă astfel:

10h Jh Qh Kh Ah

5s 6s 7s 8s 9s

Ah Ac Ad As 2h

As Ad Ac Kh Ks

2h 4h 6h 8h Kh

5h 6c 7d 8s 9h

Ac Ah As 2d 7c

Kd Kc Qh Qs Jd

Ac Ah 9s 8d 7c

Ah 8s 6d 6c 2h

Date. Fișierul text **2-9.txt** conține pe prima linie un număr natural n ($1 \leq n \leq 1000$) iar apoi n mâini de poker (una pe fiecare linie) numai cu cărți mici (de la 2 la 9, fără cărțile cu valorile 10, J, Q, K, A). Aceste mâini de poker sunt mai ușor de clasificat. Rezolvarea perfectă a acestui caz vă duce la nota 8.

Fișierul text **2-A.txt** conține pe prima linie un număr natural n ($1 \leq n \leq 1000$), iar apoi n mâini de poker (una pe fiecare linie) cu cărți mici (de la 2 la 9), dar și cu cărți mari având valoarea 10, J, Q, K sau A. Aceste mâini de poker sunt mai greu de clasificat. Rezolvarea perfectă a acestui caz vă duce la nota 10.

Subiectul de examen din septembrie 2018

Cerințe:

- a. Definiți structura *carte* care să permită memorarea valorii și a culorii unei cărți. **(0.5 puncte)**
- b. Scrieți o funcție cu numele *citestemaini* care citește într-o matrice alocată dinamic cu *n* linii și 5 coloane cu elemente de tip *carte* mânile dintr-un fișier text. Pe fiecare din cele *n* linii ale matricei veți stoca o mână reprezentată de cele 5 cărți de pe linia din fișierul text. Cititi fiecare mână parsând fiecare linie din fișierul text. Pentru fișierul cu cărți mici acest lucru este simplu (formatul este stabil: o cifră pentru valoare + un caracter pentru culoare), pentru fișierul cu cărți mari lucrurile sunt puțin mai grele (formatul este instabil: o cifră sau două cifre sau un caracter pentru valoare + un caracter pentru culoare). **(3.5 puncte)**
- c. Scrieți o funcție cu numele *clasificaMaini* în care aplicați regulile de clasificare a mânilor de poker pe baza ierarhiei de mai sus. Pentru fișierul cu cărți mici acest lucru este mai simplu (nu există chintă roială și cazuri extreme), iar pentru fișierul cu cărți mari lucrurile sunt puțin mai grele. Spre exemplu, putem avea o chintă formată dintr-un as și cărțile cu valorile 2, 5, 3, 4. De asemenea, mâna 8h 10h 9h Qh Jh este o chintă de culoare. **(4 puncte)**
- d. Scrieți un program care prin apeluri ale funcțiilor de mai sus afișează pe ecran clasa din care face parte fiecare mână de poker procesată. De asemenea, programul va afișa și numărul de mâini din fiecare din cele 10 clase în ordinea descrescătoare a frecvențelor lor (folosiți funcția *qsort* pentru sortare). **(1 punct)**

Atenție: ordinea cărților într-o mână nu contează. Mânile 8h 10h 9h Qh Jh și 10h 8h 9h Qh Jh sunt identice. Prin excepție, un as (A) poate lua și valoarea 1 astfel încât o mână ce conține cărțile A, 2, 3, 4, 5 este o chintă (posibil chintă de culoare dacă cărțile au aceeași culoare).

Soluție

```
solutie.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 //punctul a
6 typedef struct
7 {
8     int valoare;
9     char culoare;
10} carte;
11
12 typedef struct
13 {
14     carte** tablou;
15     int n;
16 } matriceCarti;
17
18
19 enum clasaMană{cartemare, opereche, douaperechi, treideunfel, chinta,
20     culoare, full, careu, chintadeculoare, chintaroiala};
21
```

```
int clasificaMană(carte* t)
{
    int frecValori[13] = {0}; //pentru valorile 2,3,4...,10,11=J,12=Q,13=K,14 = As
    int frecCulori[4] = {0};
    int i;
    for(i=0; i<5; i++)
    {
        frecValori[t[i].valoare-2]++;
        switch(t[i].culoare)
        {
            case 'h': frecCulori[0]++; break;
            case 'd': frecCulori[1]++; break;
            case 's': frecCulori[2]++; break;
            case 'c': frecCulori[3]++; break;
        }
    }
}
```

Soluție

```
170     //verifica mainile pe baza frecvenței vectorior si a culorilor
171
172     //chinta roiala
173     if(eCuloare(frecCulori) && eChintaMare(frecValori))
174         return chintaroiala;
175
176     //chinta de culoare
177     if(eCuloare(frecCulori) && eChinta(frecValori))
178         return chintadeculoare;
179
180     //careu
181     if(eCareu(frecValori))
182         return careu;
183
184     int nrPerechi = calculeazaNumarPerechi(frecValori);
185     //full
186     if(eTrei(frecValori) && (nrPerechi==1))
187         return full;
188
189     //culoare
190     if(eCuloare(frecCulori))
191         return culoare;
192
193     //chinta
194     if(eChinta(frecValori))
195         return chinta;
196
197     //treideunfel
198     if(eTrei(frecValori))
199         return treideunfel;
200
201     //doua perechi
202     if(nrPerechi==2)
203         return douaperechi;
204
205     //o pereche
206     if(nrPerechi==1)
207         return opereche;
208
209     return cartemare;
210 }
```

Soluție

```
88 int eCuloare(int c[]) //v aici e vectorul de frecvențe ale culorilor
89 {
90     int i;
91     for(i=0;i<4;i++)
92         if (c[i]==5)
93             return 1;
94     return 0;
95 }
96
97 int eChinta(int v[]) //v aici e vectorul de frecevnta ale valorilor cu 13 componente
98 {
99     //verificam 2-6, 3-7,..., 9-14 si apoi cazul extrem 2,3,4,5,13(As)
100    int i, j;
101    for(i=0;i<9;i++)
102        if ((v[i]==1) && (v[i+1]==1) && (v[i+2]==1) && (v[i+3]==1) && (v[i+4]==1))
103            return 1;
104    //cazul extrem
105    if ((v[0]==1) && (v[1]==1) && (v[2]==1) && (v[3]==1) && (v[12]==1))
106        return 1;
107
108    //altfel nu e chinta
109    return 0;
110 }
111
112 int eChintaMare(int v[])
113 {
114     //verificam 9-13
115     if ((v[8]==1) && (v[9]==1) && (v[10]==1) && (v[11]==1) && (v[12]==1))
116         return 1;
117     //altfel nu e chinta mare
118     return 0;
119 }
120
121 int eCareu(int v[])
122 {
123     int i;
124     for(i=0;i<13;i++)
125         if(v[i]==4)
126             return 1;
127     return 0;
128 }
```

Cuprinsul cursului de azi

1. Tipuri derivate de date: tablouri, siruri de caractere, structuri, campuri de biti, uniuni, enumerari, tipuri definite de utilizator (typedef).
2. Instructiuni de control.
3. Directive de procesare. Macrodefinitii.

Cuprinsul cursului de azi

1. Tipuri derivate de date: tablouri, siruri de caractere, structuri, campuri de biti, uniuni, enumerari, tipuri definite de utilizator (typedef).
2. Instructiuni de control.
3. Directive de procesare. Macrodefinitii.

Instrucțiuni de control

- ❑ reprezintă:
 - ❑ elementele fundamentale ale funcțiilor
 - ❑ comenzi date mașinii
 - ❑ determină fluxul de control al programului (ordinea de execuție a operațiilor din program)
- ❑ **instrucțiuni de bază**
 - ❑ instrucțiunea expresie
 - ❑ instrucțiunea vidă
 - ❑ instrucțiuni sevențiale/liniare
 - ❑ instrucțiuni decizionale/selective simple sau multiple
 - ❑ instrucțiuni repetitive/ciclice/iterative
 - ❑ instrucțiuni de salt condiționat/necondiționat
 - ❑ instrucțiunea return

Instructiuni de control

- ❑ programare structurată
 - ❑ Teorema Böhm-Jacopini: fluxul de control poate fi exprimat folosind doar trei tipuri de **instructiuni de control**:
 - ❑ instructiuni secentiale
 - ❑ instructiuni decizionale
 - ❑ instructiuni repetitive

Instructiunea expresie

- ❑ formată dintr-o expresie urmată de semnul ;
 - ❑ expresie;
- ❑ cele mai frecvente
 - ❑ se bazează pe expresii de atribuire, aritmetice și de incrementare / decrementare
 - ❑ adică expresii care au efecte secundare: schimbă valoarea unui operand

Exemple:

```
a = 123;  
b = a + 5;  
b++;
```

- ❑ expresie vs. instructiune

Expresie

i++

a=a-5

Instructiune

i++;

a=a-5;

Instrucțiunea vidă

- ❑ o instrucțiune care constă doar din caracterul ;
 - ❑ folosită în locurile în care limbajul impune existența unei instrucțiuni, dar programul nu trebuie să execute nimic
- ❑ cel mai adesea instrucțiunea vidă apare în combinație cu instrucțiunile repetitive
 - ❑ vezi instrucțiunea for

Instrucțiunea compusă

- ❑ numită și instrucțiune bloc
- ❑ alcătuită prin gruparea mai multor instrucțiuni și declarații
 - ❑ folosite în locurile în care sintaxa limbajului presupune o singură instrucțiune, dar programul trebuie să efectueze mai multe instrucțiuni
 - ❑ gruparea
 - ❑ includerea instrucțiunilor între accolade, {}
 - ❑ astfel compilatorul va trata secvența de instrucțiuni ca pe o singură instrucțiune
 - ❑ *{secvență de declarații și instrucțiuni }*

Instrucțiuni decizionale/selective

- ❑ ramifică fluxul de control în funcție de valoarea de adevăr a expresiei evaluate

- ❑ limbajul C furnizează două instrucțiuni decizionale
 - ❑ instrucțiunea **if** – instrucțiune decizională simplă
 - ❑ instrucțiunea **switch** - instrucțiune decizională multiplă

Instrucțiunea IF

- ❑ instrucțiunea selectivă fundamentală
 - ❑ permite selectarea uneia dintre două alternative în funcție de valoarea de adevăr a expresiei testate
- ❑ forma generală:

```
if (expresie)
    {bloc de instructiuni 1};
else
    {bloc de instructiuni 2};
```
- ❑ valoarea expresiei incluse între paranteze rotunde trebuie să fie un scalar
 - ❑ dacă e nenulă se execută *blocul de instrucțiuni 1 (instrucțiunea compusă)*, altfel se execută *blocul de instrucțiuni 2*
- ❑ ramura **else** poate lipsi

Instrucțiunea IF

- se citesc numerele naturale a și b de la tastatură. Să se afișeze ultima cifră a numărului a^b .

```
seminar1.c
1 #include <stdio.h>
2 #include <math.h>
3
4 int main()
5 {
6     int a,b;
7     scanf("%d %d",&a,&b);
8     if (a==0)
9     {
10         if (b==0)
11         {
12             printf("0 la puterea 0, caz de nedeterminare \n");
13             return 0;
14         }
15     }
16
17     if(b==0)
18     {
19         printf("1\n");
20         return 0;
21     }
22
23     printf("%d \n", (int)pow(a%10,b%4+4) % 10);
24
25     return 0;
```

Instructiunea IF

- erori frecvente:
 - nu includem acoladele

```
if (a>b)
    a = a+b;
    b = a;
```

- Întotdeauna se va executa instrucțiunea `b=a;`
- confundarea operatorul de egalitate `==`, cu operatorul de atribuire `=`

Exemple comparative:

```
a = 2;
if ( a == 10 )
    printf("a este 10 \n");
```

```
a = 2;
if ( a = 10 )
    printf("a este 10 \n");
```

- mesajul `a este 10` nu va fi afișat
 - după testarea egalității folosind operatorul `==` se returnează 0
 - (2 nefiind egal cu 10)
- mesajul `a este 10` va fi afișat întotdeauna
 - expresia `a = 10`
 - a ia valoarea 10
 - se evaluatează adevărat

Instrucțiunea IF

- instrucțiuni IF imbricate
 - pe oricare ramură poate conține alte instrucțiuni if
- forma generală:

```
if (expresie1)
    if (expresie2) {bloc de instructiuni 1};
    else {bloc de instructiuni 2};
else
    {bloc de instructiuni 2};
```

Exemplu:

```
int a, b;
// ...
if ( a <= b )
    if ( a == b )
        printf("a = b");
    else
        printf("a < b");
else
    printf("a > b");
```

Instructiunea IF

- instructiuni IF în cascadă
 - testează succesiv mai multe condiții implementând o variantă de selecție multiplă
- forma generală:

```
if (expresie1) {bloc de instructiuni 1};  
else if (expresie2) {bloc de instructiuni 2};  
else if (expresie3) {bloc de instructiuni 3};  
...  
else {bloc de instructiuni N};
```

```
#include <stdio.h>  
  
int main() {  
    float nota;  
  
    printf("Introduceti o nota in intervalul [1, 10]: ");  
    scanf("%f", &nota);  
  
    if ( nota > 9 && nota <= 10)  
        printf("Calificativul este: EXCELENȚA");  
    else if ( nota > 8 && nota <= 9)  
        printf("Calificativul este: Foarte bine");  
    else if ( nota > 7 && nota <= 8)  
        printf("Calificativul este: Bine");  
    else if ( nota > 5 && nota <= 7)  
        printf("Calificativul este: Suficient");  
    else printf("Calificativul este: Insuficient");  
  
    return 0;  
}
```

Instrucțiunea SWITCH

- ❑ efectuează selecția multiplă
 - ❑ util când expresia de evaluat are mai multe valori posibile
- ❑ forma generală

```
switch (expresie){  
    case val_const_1: {bloc de instructiuni 1};  
    case val_const_2: {bloc de instructiuni 2};  
    ....  
    case val_const_n: {bloc de instructiuni N};  
    default: {bloc de instructiuni D};  
}
```

Instrucțiunea SWITCH

- ❑ poate fi întotdeauna reprezentată prin instrucțiunea IF
 - ❑ de regulă prin instrucțiuni IF în cascadă
- ❑ În cazul instrucțiunii switch fluxul de controlul sare direct la instrucțiunea corespunzătoare valorii expresiei testate
- ❑ switch este mai rapid și codul rezultat mai ușor de înțeles

Instrucțiunea SWITCH

- se citesc numerele naturale a și b de la tastatură. Să se afișeze ultima cifră a numărului a^b .

```
seminar1_2.c
01 #include <stdio.h>
02 #include <math.h>
03
04 int main()
05 {
06     int a,b;
07     scanf("%d %d",&a,&b);
08     if (a==0)
09     {
10         if (b==0)
11         {
12             printf("0 la puterea 0, caz de nedeterminare \n");
13             return 0;
14         }
15     }
16     if(b==0)
17     {
18         printf("1\n");
19         return 0;
20     }
21
22     a = a%10;
23     switch (b%4)
24     {
25         case 0: printf("%d\n",a*a*a*a % 10); break;
26         case 1: printf("%d\n",a); break;
27         case 2: printf("%d\n",a*a % 10); break;
28         case 3: printf("%d\n",a*a*a % 10); break;
29     }
30     return 0;
31 }
```

Instrucțiunea SWITCH

- ❑ efectuează selecția multiplă
 - ❑ util când expresia de evaluat are mai multe valori posibile
- ❑ forma generală

```
switch (expresie){  
    case val_const_1: bloc de instructiuni 1;  
    case val_const_2: bloc de instructiuni 2;  
    ....  
    case val_const_n: bloc de instructiuni N;  
    default: bloc de instructiuni D;  
}
```

Instrucțiunea SWITCH

- ❑ mod de funcționare și constrângeri:
 - ❑ expresie se evaluează o singură dată la intrarea în instrucțiunea switch
 - ❑ expresie trebuie să rezulte într-o valoare întreagă (poate fi inclusiv caracter, dar nu valori reale sau siruri de caractere)
 - ❑ valorile din ramurile **case** notate **val_ const_i** (numite și etichete) trebuie să fie constante întregi (sau caracter), reprezentând o singură valoare
 - ❑ nu se poate reprezenta un interval de valori
 - ❑ instrucțiunile care urmează după etichetele **case** nu trebuie incluse între accolade, deși pot fi mai multe instrucțiuni, iar ultima instrucțiune este de regulă instrucțiunea **break**

Instrucțiunea SWITCH

- mod de funcționare și constrângeri:
 - dacă valoarea expresiei se potrivește cu vreuna din valorile constante din ramurile case, atunci se vor executa instrucțiunile corespunzătoare acelei ramuri, altfel se execută instrucțiunea de pe ramura **default** (dacă există)
 - dacă nu s-a întâlnit **break** la finalul instrucțiunilor de pe ramura pe care s-a intrat, atunci se continuă execuția instrucțiunilor de pe ramurile consecutive (fără verificarea etichetei) până când se ajunge la **break** sau la sfârșitul instrucțiunii **switch**, moment în care se ieșe din instrucțiunea **switch** și se trece la execuția instrucțiunii imediat următoare
 - ramura **default** este opțională iar poziția relativă a acesteia printre celelalte ramuri nu este relevantă
 - dacă nici o etichetă nu se potrivește cu valoarea expresiei testate și nu există ramura **default**, atunci instrucțiunea **switch** nu are nici un efect

Instrucțiunea SWITCH

- omiterea instrucțiunii break de la finalul unei ramuri case
 - accidentală - este o eroare frecventă
 - deliberată - permite fluxului de execuție să intre și pe ramura case următoare

Instrucțiunea SWITCH

```
seminar1_3.c      x

1 #include <stdio.h>
2 #include <math.h>
3
4 int main()
5 {
6     int a,b;
7     scanf("%d %d",&a,&b);
8     if (a==0)
9     {
10         if (b==0)
11         {
12             printf("0 la puterea 0, caz de nedeterminare \n");
13             return 0;
14         }
15     }
16     if(b==0)
17     {
18         printf("1\n");
19         return 0;
20     }
21
22     a = a%10;
23     switch (b%4)
24     {
25     case 0: printf("%d\n",a*a*a*a % 10);
26     case 1: printf("%d\n",a);
27     case 2: printf("%d\n",a*a % 10);
28     case 3: printf("%d\n",a*a*a % 10);
29     }
30     return 0;
31 }
32 }
```

```
Bogdan-Alexes-MacBook-Pro:curs4 bogdan$ gcc seminar1_3.c
Bogdan-Alexes-MacBook-Pro:curs4 bogdan$ ./a.out
12 33
2
4
8
```

Instrucțiuni repetitive

- ❑ sunt numite și instrucțiuni iterative sau ciclice
- ❑ efectuează o serie de instrucțiuni în mod repetat fiind condiționate de o expresie de control care este evaluată la fiecare iterare
- ❑ instrucțiunile iterative furnizate de limbajul C sunt:
 - ❑ instrucțiunea repetitivă cu testare inițială **while**
 - ❑ instrucțiunea repetitivă cu testare finală **do-while**
 - ❑ instrucțiunea repetitivă cu testare inițială **for**

Instrucțiunea WHILE

- ❑ execută în mod repetat o instrucțiune atât timp cât expresia de control este evaluată la adevărat
- ❑ evaluarea se efectuează la începutul instrucțiunii
 - ❑ dacă rezultatul corespunde valorii logice adevărat
 - ❑ se execută corpului instrucțiunii, după care se revine la testarea expresiei de control
 - ❑ acești pași se repetă până când expresia va fi evaluată la fals
 - ❑ acesta va determina ieșirea din instrucțiune și trecerea la instrucțiunea imediat următoare
- ❑ forma generală: **while** (*expresie*) {bloc de instrucțiuni}

Instrucțiunea WHILE

```
unsigned int i=3;  
while (i>=0){  
    printf("%d\n",i);  
    i--;  
}
```

Ce afișează sevența de cod alăturată?

CICLEAZA!!!

Instrucțiunea DO-WHILE

- ❑ efectuează în mod repetat o instrucțiune atât timp cât expresia de control este adevărată
- ❑ evaluarea se face la finalul fiecărei iterații
 - ❑ corpul instrucțiunii este executat cel puțin o dată
- ❑ forma generală: **do** {bloc de instrucțiuni} **while** (expresie);
- ❑ eroare frecventă: omiterea caracterului punct și virgulă de la finalul instrucțiunii

Instrucțiunea FOR

- ❑ evaluarea expresiei de control se face la începutul fiecărei iterații

- ❑ forma generală:

```
for (expresii_init; expresie_control; expresie_ajustare)
    {bloc de instructiuni}
```

- ❑ poate fi întotdeauna transcrisă folosind o instrucțiune while:

```
expresii_init;
while (expresie_control)
    {bloc de instructiuni
expresii_ajustare;}
```

Instrucțiunea FOR

```
// insumarea elementelor din vectorul de intregi cu for  
  
for (i = 0, suma = 0; i < nr ; i++)  
{  
    printf("v[%d]: ", i);  
    scanf("%d", &v[i]);  
    suma += v[i];  
}
```

- instrucțiunea for permite ca elementul de ajustare din antetul instrucțiunii să cuprindă mai multe expresii
 - se poate ajunge chiar și la situația în care corpul instrucțiunii nu mai conține nici o instrucțiune de executat
 - se folosește **instrucțiunea vidă** (punct și virgulă) pentru a indica sfârșitul instrucțiunii for

```
// insumarea elementelor din vectorul de intregi cu for  
  
for (i = 0, suma = 0; i < nr ; suma += v[i], i++);  
printf("Suma elementelor este: %d", suma);
```

Instrucțiunea FOR

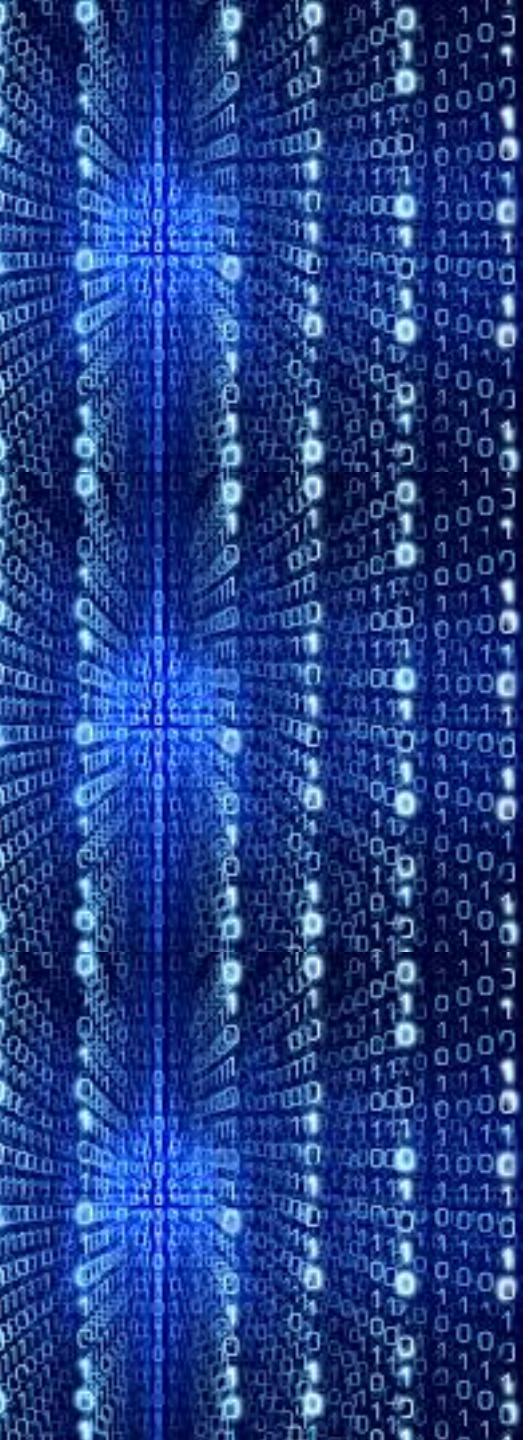
```
int i=0;  
for(; i<=5; i++);  
    printf("%d", i);
```

Ce afișează sevența de cod alăturată?

Afișează 6

Instrucțiunile break, continue și goto

- realizează **salturi**
 - îintrerup controlului secvențial al programului și continuă execuția dintr-un alt punct al programului sau chiar provoacă ieșirea din program
- instrucțiunea **break** provoacă ieșirea din instrucțiunea curentă
- instrucțiunea **continue** provoacă trecerea la iterată imediat următoare în instrucțiunea repetitivă
- instrucțiunea **goto** produce un salt la o etichetă predefinită în cadrul aceleasi funcții



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,

2018-2019

Cursul 5

Funcția qsort

- antetul funcției qsort este:

```
void qsort (void *adresa, int nr_elemente, int dimensiune_element,  
int (*cmp) (const void *, const void *))
```

- adresa = pointer la adresa primului element al tabloului ce urmează a fi sortat
(pointer generic – nu are o aritmetică precizată)
- nr_elemente = numărul de elemente al vectorului
- dimensiune_element = dimensiunea în octeți a fiecărui element al tabloului
(char = 1 octet, int = 4 octeți, etc)
- **cmp – funcția de comparare a două elemente, pointer la o funcție**

Funcția qsort

```
void qsort (void *adresa, int nr_elemente, int dimensiune_element,  
int (*cmp) (const void *, const void *))  
  
int cmp(const void *a, const void *b)
```

adresele a două elemente din tablou

Cmp este o funcție generică comparator, compară 2 elemente de orice tip din tablou. Întoarce:

- un număr < 0 dacă vrem elementul de la adresa **a** la stânga (înaintea) elementului de la adresa **b**
- un număr > 0 dacă vrem elementul de la adresa **a** la dreapta (după) elementului de la adresa **b**
- 0, dacă nu contează

Funcția qsort pentru întregi

```
void qsort (void *adresa, int nr_elemente, int dimensiune_element, int  
(*cmp) (const void *, const void *))
```

Exemplu de funcție cmp pentru sortarea unui vector de numere întregi:

```
int cmp(const void* a, const void *b)  
{  
    int va, vb;  
    va = *(int*)a;  
    vb = *(int*)b;  
    if(va < vb) return -1;  
    if(va > vb) return 1;  
    return 0;  
}
```



```
int cmp(const void* a, const void *b)  
{  
    return *(int*)a - *(int*)b;  
}
```

Funcția qsort pentru întregi

```
exempluqsort.c
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int cmp(const void* a, const void *b)
5 {
6     return *(int*)a - *(int*)b;
7 }
8
9 int main()
10 {
11     int v[] = {0, 5, -6, 9, 7, 12, 8, 7, 4};
12     qsort(v, 9, sizeof(int), cmp);
13     for( int i = 0; i < sizeof(v)/sizeof(int); i++)
14         printf("%d \t", v[i]);
15     printf("\n");
16     return 0;
17 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs5 bogdan$ ./a.out
```

```
-6      0      4      5      7      7      8      9      12
```

Funcția qsort pentru structuri

Exercițiu seminar: avem o structură care reține numele, prețul, cantitatea pentru fiecare produs dintr-un magazin. Se dă un vector de asemenea produse pe care vreau să îl sorteze descrescător după preț și în caz de prețuri egale în ordinea alfabetică a numelor produselor.

```
produs.c

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 struct Produs
6 {
7     char nume[25];
8     double pret;
9     double cantitate;
10 };
11
12
13 int cmpProduse(const void *a, const void* b)
14 {
15     if (((struct Produs *)a)->pret == ((struct Produs *)b)->pret)
16         return strcmp(((struct Produs *) a)->nume,((struct Produs *)b)->nume);
17
18     if (((struct Produs *)a)->pret > ((struct Produs *)b)->pret)
19         return -1;
20     return +1;
21 }
22 }
```

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, siruri 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)

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

□ Tablouri și pointeri

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

□ Siruri 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.
- Programare generică.

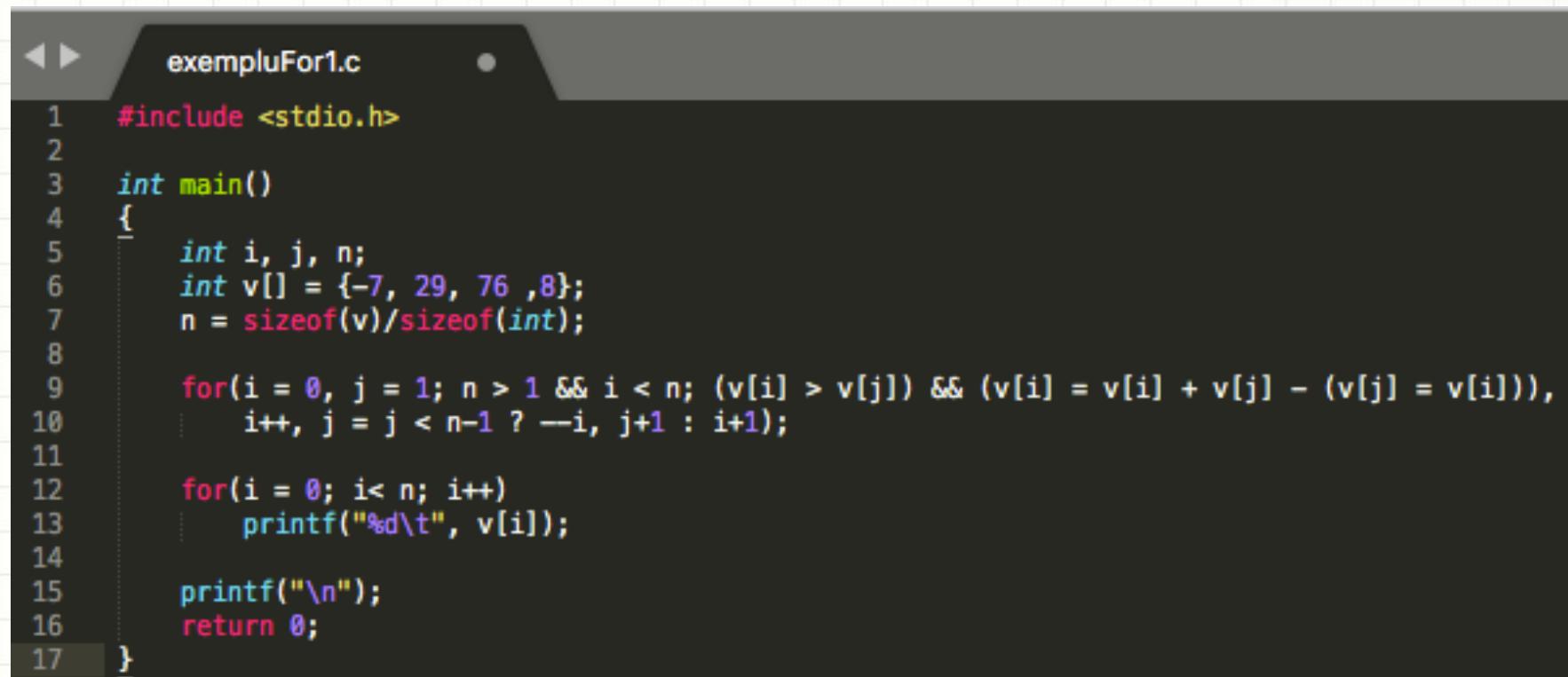
□ Recursivitate

Cuprinsul cursului de azi

1. Instrucțiuni de control (break, continue, goto, return).
2. Etapele realizării unui program C.
3. Directive de preprocessare. Macrodefiniții.
4. Funcții de citire/scriere.

Exemplu FOR

- putem scrie un program într-un singur for;



The screenshot shows a code editor window with the file 'exempluFor1.c' open. The code is a C program that prints the elements of an array in descending order. It includes #include <stdio.h>, a main function with variable declarations, a nested for loop for sorting, and a printf statement for output.

```
#include <stdio.h>
int main()
{
    int i, j, n;
    int v[] = {-7, 29, 76, 8};
    n = sizeof(v)/sizeof(int);

    for(i = 0, j = 1; n > 1 && i < n; (v[i] > v[j]) && (v[i] = v[i] + v[j] - (v[j] = v[i])), i++, j = j < n-1 ? -i, j+1 : i+1);
    for(i = 0; i < n; i++)
        printf("%d\t", v[i]);
    printf("\n");
    return 0;
}
```

[Bogdan-Alexes-MacBook-Pro:curs5 bogdan\$ gcc exempluFor1.c

[Bogdan-Alexes-MacBook-Pro:curs5 bogdan\$./a.out

-7 8 29 76

Exemple FOR

exempluFor2.c

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int i, j, n, r, s, w;
6     int v[100];
7
8     for(i = 0, j = 1, r = s = w = 0, printf("Numarul de elemente: "), scanf("%d", &n), printf("\nElementele:\n");
9         i < n ? 1 : (r == 0 ? (i = 0, r = 1) : (s == 0 ? (i = 0, s = 1,
10             printf("\nTabloul sortat:\n")) : (w == 0 ? 1 : (printf("\n"), 0))));
11         (r == 0) && (printf("v[%d] = ", i), scanf("%d", &v[i])),
12         (n > 1) && (r == 1) && (s == 0) && (v[i] > v[j]) && (v[i] = v[i] + v[j] - (v[j] = v[i])),
13         (r == 1) && (s == 1) && (w == 0) && (printf("%d ", v[i])),
14         i++, (i == n) && (s == 1) && (w == 0) && (w = 1),
15         (r == 1) && (s == 0) && (i < n-1) && (j = j < n-1 ? --i, j+1 : i+1));
16
17     return 0;
18 }
```

Exemple FOR

```
exempluFor2.c ●

1 #include <stdio.h>
2
3 int main()
4 {
5     int i, j, n, r, s, w;
6     int v[100];
7
8     for(i = 0, j = 1, r = s = w = 0, printf("Numarul de elemente: "), scanf("%d", &n), printf("\nElementele:\n");
9         i < n ? 1 : (r == 0 ? (i = 0, r = 1) : (s == 0 ? (i = 0, s = 1,
10             printf("\nTabloul sortat:\n")) : (w == 0 ? 1 : (printf("\n"), 0))));
11         (r == 0) && (printf("v[%d] = ", i), scanf("%d", &v[i])),
12         (n > 1) && (r == 1) && (s == 0) && (v[i] > v[j]) && (v[i] = v[i] + v[j] - (v[j] = v[i])),
13         (r == 1) && (s == 1) && (w == 0) && (printf("%d ", v[i])),
14         i++, (i == n) && (s == 1) && (w == 0) && (w = 1),
15         (r == 1) && (s == 0) && (i < n-1) && (j = j < n-1 ? --i, j+1 : i+1));
16
17     return 0;
18 }
```

Bogdan-Alexes-MacBook-Pro:curs5 bogdan\$./a.out

```
Numarul de elemente: 5

Elementele:
v[0] = 10
v[1] = 11
v[2] = 13
v[3] = 14
v[4] = 12

Tabloul sortat:
10 11 12 13 14
```

Instrucțiunile break, continue și goto

- realizează **salturi**
 - îintrerup controlul secvențial al programului și continuă execuția dintr-un alt punct al programului sau chiar provoacă ieșirea din program
- instrucțiunea **break** provoacă ieșirea din instrucțiunea curentă
- instrucțiunea **continue** provoacă trecerea la iterată imediat următoare în instrucțiunea repetitivă
- instrucțiunea **goto** produce un salt la o etichetă predefinită în cadrul aceleasi funcții

Instrucțiunea goto

- instrucțiunea **goto** produce un salt la o etichetă predefinită în cadrul aceleiași funcții
- forma generală: **goto eticheta**
 - unde eticheta este definită în program
 - eticheta: instructiune

```
int i=0;  
eticheta:  
    if(i%3!=0)  
        printf("i=%d\n",i);  
    i++;  
    if(i<10)  
        goto eticheta;  
return 0;
```

i=1
i=2
i=4
i=5
i=7
i=8

Instrucțiunile break, continue și goto

```
//Insumarea tuturor numerelor prime
dintr-un vector de intregi, pana la
intalnirea primului numar multiplu de
100
#include <stdio.h>
#include <math.h>
int main()
{
    int v[]={640,2,29,1,49,
              33,23,800,47,3};
    int suma=0;
    int i;
    int nr=sizeof(v)/sizeof(int);
    for (i=0; i<nr; i++)
    {
        if (v[i]%100==0)
            goto afisare_suma;
        if (v[i]<2)
            continue;
```

```
        int prim=1;
        int k;
        double epsilon=0.001;
        int limit= (int) (sqrt(v
[i])+epsilon);
        for (k=2; k<=limit; k++)
            if (v[i]%k==0)
                {
                    prim=0;
                    break;
                }
        if (prim)
            suma+=v[i];
    }
afisare_suma:
    printf("Suma este %d", suma);
    return 0;
}
```

Instrucțiunile break, continue și goto

```
//Acceasi problema dar fara a
utiliza break, continue si goto
#include <stdio.h>
#include <math.h>
int main()
{
    int v[]={640,2,29,1,49,
              33,23,800,47,3};
    int suma=0;
    int i=0;
    int nr=sizeof(v) / sizeof(int);
    while (i<nr && v[i]%100!=0)
    {
        if (v[i]>=2)
        {
            int prim=1;
            int k=2;
            double epsilon=0.001;
```

```
        int limit= (int) (sqrt(v[i]))
                  +epsilon);
        while (prim && k<=limit)
        {
            if (v[i]%k==0)
                prim=0;
            k++;
        }
        if (prim)
            suma+=v[i];
        } i+
        ;
    }
    printf("Suma este %d", suma);
    return 0;
```

Instrucțiunea RETURN

- ❑ se folosește pentru a returna fluxul de control al programului apelant dintr-o funcție (main sau altă funcție)

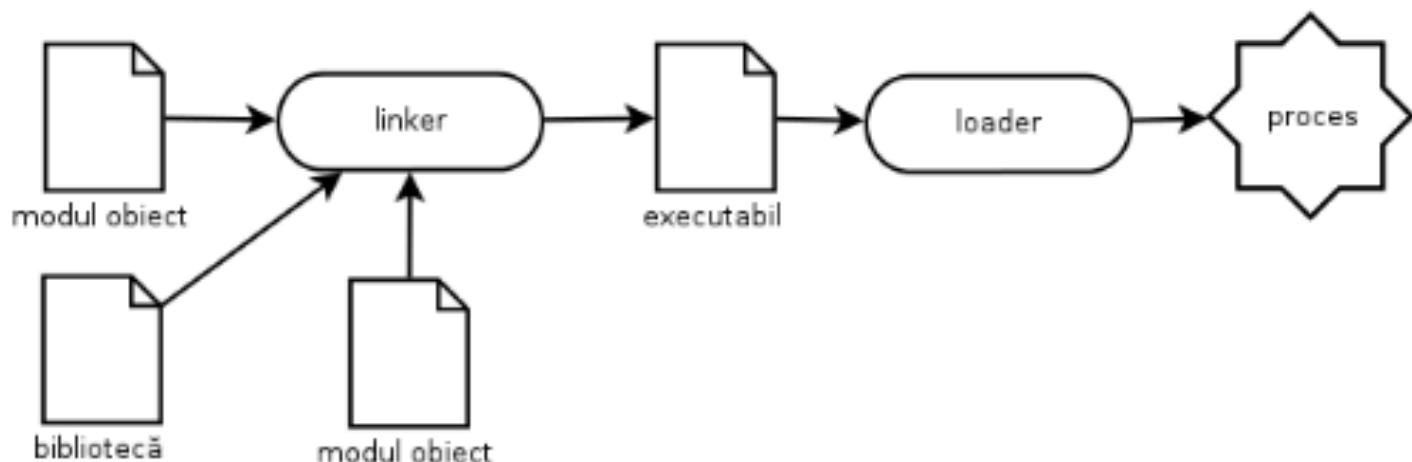
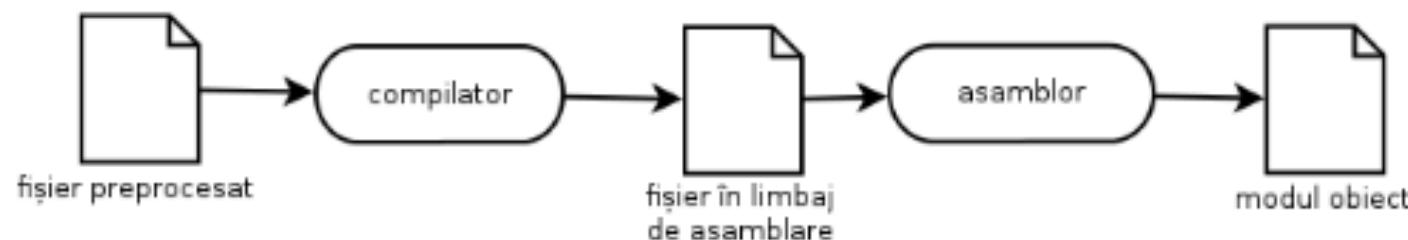
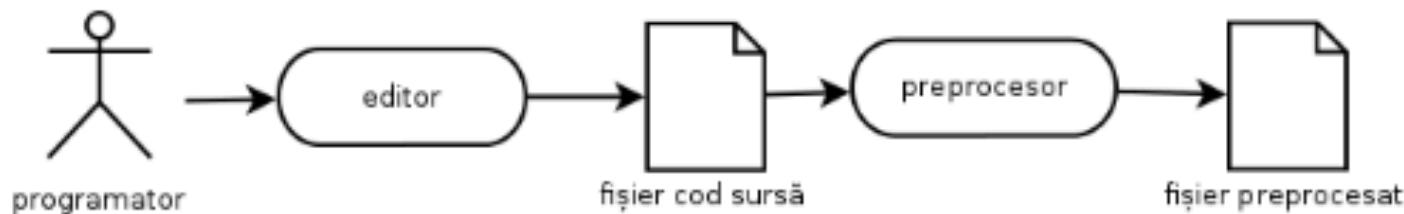
- ❑ are două forme:
 - ❑ `return;`
 - ❑ `return expresie;`

```
seminar1.c
1 #include <stdio.h>
2 #include <math.h>
3
4 int main()
5 {
6     int a,b;
7     scanf("%d %d",&a,&b);
8     if (a==0)
9     {
10         if (b==0)
11         {
12             printf("0 la puterea 0, caz de nedeterminare \n");
13             return 0;
14         }
15     }
16
17     if(b==0)
18     {
19         printf("1\n");
20         return 0;
21     }
22
23     printf("%d \n", (int)pow(a%10,b%4+4) % 10);
24
25     return 0;
```

Cuprinsul cursului de azi

1. Instrucțiuni de control (break, continue, goto, return).
2. Etapele realizării unui program C.
3. Directive de preprocessare. Macrodefiniții.
4. Funcții de citire/scriere.

Etapele realizării unui program în C



Etapele realizării unui program în C

- ❑ se parcurg următoarele etape pentru obținerea unui cod executabil:
 - ❑ **editarea** codului sursă
 - ❑ salvarea fișierului cu extensia .c
 - ❑ **preprocesarea**
 - ❑ efectuarea directivelor de preprocesare (**#include**, **#define**)
 - ❑ ca un editor – modifică și adaugă la codul sursă
 - ❑ **compilarea**
 - ❑ verificarea sintaxei
 - ❑ codul este tradus din cod de nivel înalt în limbaj de asamblare
 - ❑ **asamblarea**
 - ❑ transformare în cod obiect (limbaj mașină) cu extensia .o, .obj
 - ❑ nu este încă executabil !
 - ❑ **link-editarea** (editarea legăturilor)
 - ❑ combinarea codului obiect cu alte coduri obiect (al bibliotecilor asociate fișierelor header)
 - ❑ transformarea adreselor simbolice în adrese reale

Etapele realizării unui program în C

□ Exemplul 1

```
exemplu1.c

1 // program "exemplu1.c" scris de Bogdan Alexe
2 // ultima versiune 30.10.2018
3
4 #include <stdio.h>
5 #include "algebra.c"
6 #define MIN 0
7
8 int main()
9 {
10     int a, b;
11     do
12     {
13         printf("a=");
14         scanf("%d",&a);
15         printf("b=");
16         scanf("%d",&b);
17     } while (a<=MIN || b<=MIN);
18
19     printf("cmmdc(%d,%d) = %d \n", a,b,cmmdc(a,b));
20     return 0;
21 }
```

```
algebra.c

1 int cmmdc(int a, int b)
2 {
3     int c = a%b;
4     while (c)
5     {
6         a = b;
7         b = c;
8         c = a%b;
9     }
10 }
```

DEMO

Etapele realizării unui program în C

- din linia de comandă (pe Mac):

- **preprocesarea**

- **gcc –E exemplu1.c**

```
extern int __vsnprintf_chk (char * restrict, size_t, int, size_t,
    const char * restrict, va_list);
# 499 "/usr/include/stdio.h" 2 3 4
# 5 "exemplu1.c" 2
# 1 "./algebra.c" 1
int cmmdc(int a, int b)
{
    int c = a%b;
    while (c)
        {a = b;
         b = c;
         c = a%b;
        }
    return b;
}
# 6 "exemplu1.c" 2

int main()
{
    int a, b;
    do
    {
        printf("a=");
        scanf("%d",&a);
        printf("b=");
        scanf("%d",&b);
    } while (a<=0 || b<=0);

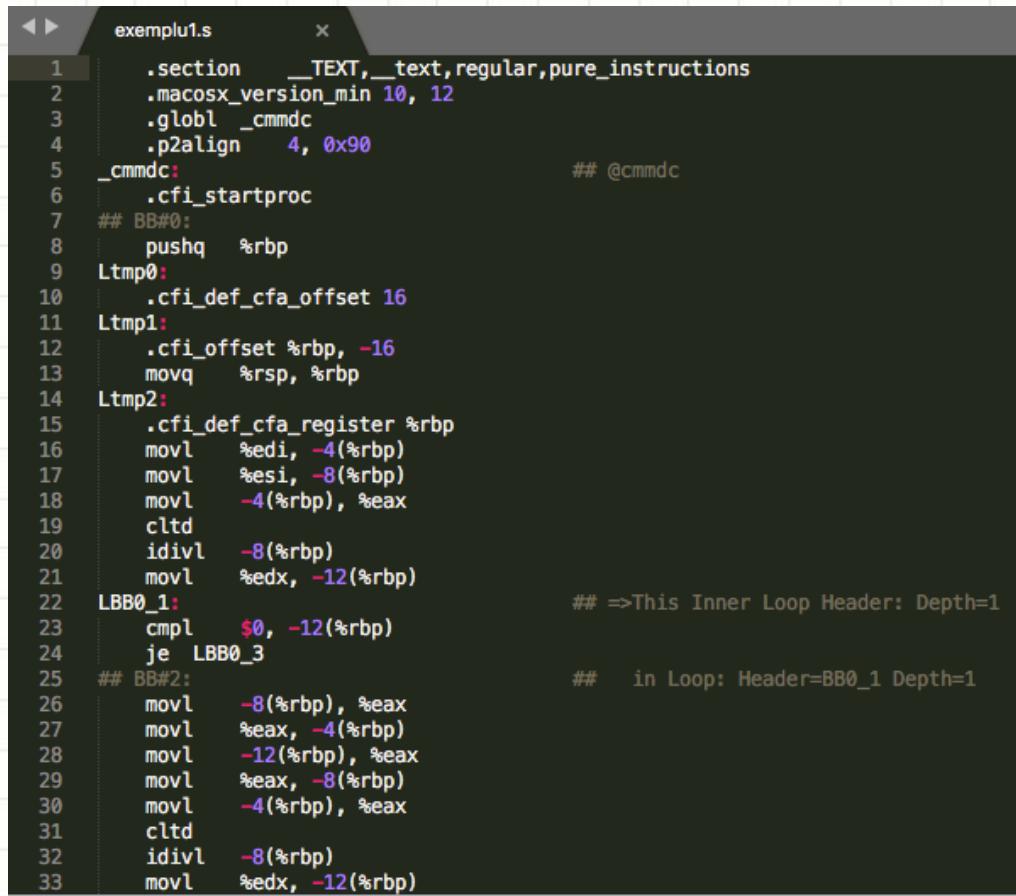
    printf("cmmdc(%d,%d) = %d \n", a,b,cmmdc(a,b));
    return 0;
}
```

Etapele realizării unui program în C

- din linia de comandă (pe Mac):

- **compilarea**

- gcc –S exemplu1.c (produce exemplu1.s)



```
exemplu1.s      x
1 .section    __TEXT,__text,regular,pure_instructions
2 .macosx_version_min 10, 12
3 .globl _cmmdc
4 .p2align 4, 0x90
5 _cmmdc:          ## @cmmdc
6     .cfi_startproc
7 ## BB#0:
8     pushq %rbp
9 Ltmp0:
10    .cfi_def_cfa_offset 16
11 Ltmp1:
12    .cfi_offset %rbp, -16
13    movq %rsp, %rbp
14 Ltmp2:
15    .cfi_def_cfa_register %rbp
16    movl %edi, -4(%rbp)
17    movl %esi, -8(%rbp)
18    movl -4(%rbp), %eax
19    cltd
20    idivl -8(%rbp)
21    movl %edx, -12(%rbp)
22 LBB0_1:          ## =>This Inner Loop Header: Depth=1
23    cmpl $0, -12(%rbp)
24    je LBB0_3
25 ## BB#2:          ## in Loop: Header=BB0_1 Depth=1
26    movl -8(%rbp), %eax
27    movl %eax, -4(%rbp)
28    movl -12(%rbp), %eax
29    movl %eax, -8(%rbp)
30    movl -4(%rbp), %eax
31    cltd
32    idivl -8(%rbp)
33    movl %edx, -12(%rbp)
```

Etapele realizării unui program în C

- din linia de comandă (pe Mac):

- **compilarea**

- gcc –S exemplu1.c (produce exemplu1.s)

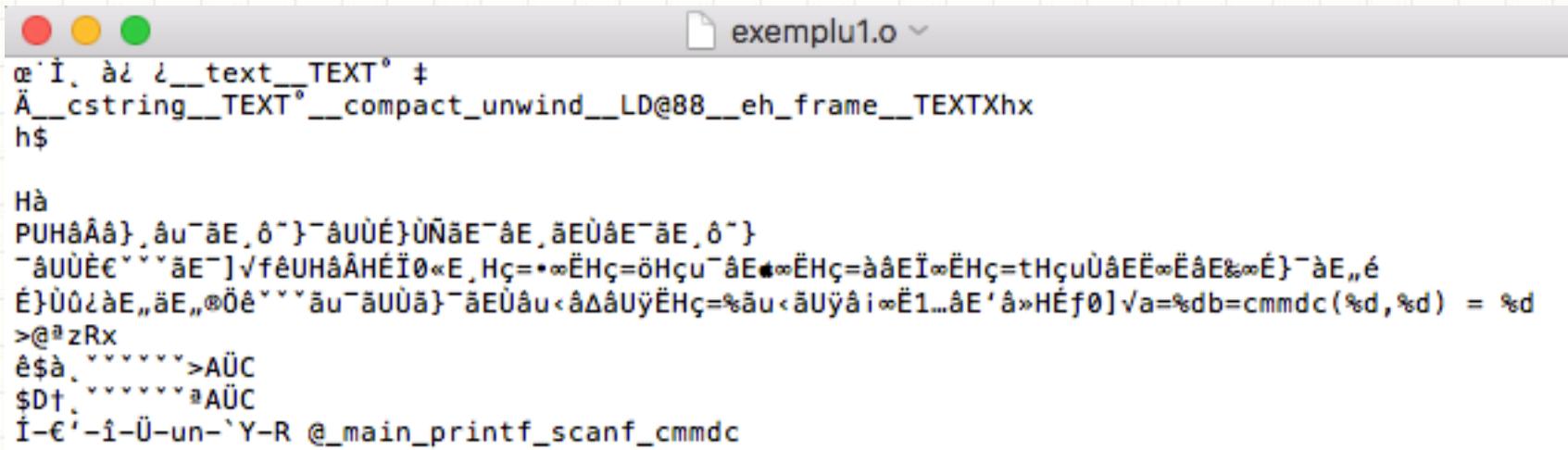
```
35 LBB0_3:  
36     movl    -8(%rbp), %eax  
37     popq    %rbp  
38     retq  
39     .cfi_endproc  
40  
41     .globl  _main  
42     .p2align 4, 0x90  
43 _main:                      ## @main  
44     .cfi_startproc  
45     ## BB#0:  
46     pushq   %rbp  
47 Ltmp3:  
48     .cfi_def_cfa_offset 16  
49 Ltmp4:  
50     .cfi_offset %rbp, -16  
51     movq    %rsp, %rbp  
52 Ltmp5:  
53     .cfi_def_cfa_register %rbp  
54     subq    $48, %rsp  
55     movl    $0, -4(%rbp)  
56 LBB1_1:                      ## =>This Inner Loop Header: Depth=1  
57     leaq    L_.str(%rip), %rdi  
58     movb    $0, %al  
59     callq   _printf  
60     leaq    L_.str.1(%rip), %rdi  
61     leaq    -8(%rbp), %rsi  
62     movl    %eax, -16(%rbp)      ## 4-byte Spill  
63     movb    $0, %al  
64     callq   _scanf  
65     leaq    L_.str.2(%rip), %rdi
```

Etapele realizării unui program în C

- din linia de comandă (pe Mac):

- **asamblarea**

- gcc –c exemplu1.c (produce exemplu1.o)



The screenshot shows a terminal window titled "exemplu1.o". The window contains assembly code, which is heavily obfuscated and appears to be a mix of random characters and valid assembly instructions. It includes labels like ".text", ".cstring", and ".eh_frame", and various assembly mnemonics and registers. The code is likely the result of the GCC compiler's internal optimization and assembly generation process.

```
exemplu1.o
.Í, àž Ł__text__TEXT* #
Å__cstring__TEXT* __compact_unwind__LD@88__eh_frame__TEXTXhx
h$  
  
Hà
PUHâÂâ}, âu-äE, ô"}-âUÙÉ}ÜÑäE-âE, äEÙâE-äE, ô"}  
-âUÙÉE***äE"]\fêUHâÂHÉÏ0«E, Hç=+∞ËHç=öHç-âE*∞ËHç=àâEÏ∞ËHç=tHçUÙâEË∞ËâE‰∞É}-àE,,é  
É}ÜÙ{àE,,äE,,øÖe***äu-äUÙä}-äEÙâu<âΔâUÿËHç=%äu<äUÿâ i∞Ë1...âE'â»HÉf0]\f=a=%db=cmmdc(%d,%d) = %d  
>@zRx  
é$à,*****>AÜC  
$D†,*****@AÜC  
í-€:-i-Ü-un-`Y-R @_main_printf_scandc
```

Etapele realizării unui program în C

- din linia de comandă (pe Mac):
 - **preprocesare + compilare + asamblare + link-editare**
 - gcc exemplu1.c
 - produce a.out ca fisier executabil
 - gcc exemplu1.c –o exemplu1Executabil
 - produce exemplu1Executabil ca fisier executabil
- În exemplul 1 se copiază funcția cmmdc în fisierul sursă exemplu1.c și apoi se compilează întreg fisierul sursă obținut
- pentru proiecte mari (zeci de mii de linii de cod) dacă schimbăm o funcție nu vrem să recompilăm întreg proiectul ci doar să compilăm fisierul cu funcția schimbată. Link-editorul va produce codul obiect final.
- principiul de programare modulară

Etapele realizării unui program în C

□ Exemplul 2

```
00 exemplu2.c x exemplu1.c x
01
02 // program "exemplu2.c" scris de Bogdan Alexe
03 // ultima versiune 30.10.2018
04
05 #include <stdio.h>
06 //#include "algebra.c"
07 #define MIN 0
08
09 int cmmdc(int,int);
10
11 int main()
12 {
13     int a, b;
14     do
15     {
16         printf("a="); scanf("%d",&a);
17         printf("b="); scanf("%d",&b);
18     } while (a<=MIN || b<=MIN);
19
20     printf("cmmdc(%d,%d) = %d \n", a,b,cmmdc(a,b));
21     return 0;
22 }
```

```
00 algebra.c x
01
02 int cmmdc(int a, int b)
03 {
04     int c = a%b;
05     while (c)
06     {
07         a = b;
08         b = c;
09         c = a%b;
10     }
11     return b;
12 }
```

DEMO

Etapele realizării unui program în C

- compilez fiecare modul în parte (“exemplu2.c”, “algebra.c”)
 - gcc –c algebra.c
 - produce algebra.o
 - gcc –c exemplu2.c
 - produce exemplu2.o
- Link-editez codul obiect (am nevoie ca “exemplu2.c” să știe unde găsește codul de executat pentru funcția “cmmdc”)
 - gcc algebra.o exemplu2.o
 - produce fisierul executabil a.out
 - gcc algebra.o exemplu2.o –o exemplu2Executabil
 - produce fisierul executabil exemplu2Executabil

Cuprinsul cursului de azi

1. Instrucțiuni de control (break, continue, goto, return).
2. Etapele realizării unui program C.
3. Directive de preprocessare. Macrodefiniții.
4. Funcții de citire/scriere.

Preprocesare în limbajul C

- ❑ preprocesarea apare înaintea procesului de compilare a codului sursă (fișier text editat într-un editor și salvat cu extensia .c).
- ❑ preprocesarea codului sursă asigură:
 - ❑ includerea conținutului fișierelor (de obicei a fișierelor *header*)
 - ❑ definirea de macro-uri (macrodefiniții)
 - ❑ compilarea condiționată
- ❑ constă în substituirea simbolurilor din codul sursă pe baza directivelor de preprocesare
- ❑ directivele de preprocesare sunt precedate de caracterul diez #

Directiva #include

- copiază conținutul fișierului specificat în textul sursă
- `#include <nume_fisier>`
 - caută nume_fisier în directorul unde se află fișierele din librăria standard instalată odată cu compilatorul
- `#include “nume_fisier”`
 - caută nume_fisier în directorul curent

Directiva #define

```
#include <stdio.h>

//constante simbolice
#define ALPHA 30
#define BETA ALPHA+10
#define GAMMA (ALPHA+10)

//macro-uri
#define MIN(a,b) (((a)<(b))?(a):(b))
#define ABS1(x) (x<0)?-x:x
#define ABS2(x) (((x)<0)?-(x):(x))
#define INTER(tip,a,b) \
    {tip c; c=a; a=b; b=c;}
```

```
int main()
{
    int x=2*BETA;
    int y=2*GAMMA;
    printf("%d %d\n",x,y);
    int m=MIN(x,y);
    printf("%d\n",m);
    int a=ABS1(x-y);
    int b=ABS2(x-y);
    printf("%d %d\n",a,b);
    INTER(int,a,b);
    printf("%d %d\n",a,b);
    INTER(int,a,b);
    printf("%d %d\n",a,b);
    return 0;
}
```

Directiva #define

```
#include <stdio.h>

//constante simbolice
#define ALPHA 30
#define BETA ALPHA+10
#define GAMMA (ALPHA+10)

//macro-uri
#define MIN(a,b) (((a)<(b))?(a):(b))
#define ABS1(x) (x<0)?-x:x
#define ABS2(x) (((x)<0)?-(x):(x))
#define INTER(tip,a,b) \
    {tip c; c=a; a=b; b=c;}
```

```
int main()
{
    int x=2*BETA;
    int y=2*GAMMA;
    printf("%d %d\n",x,y); //70 80
    int m=MIN(x,y);
    printf("%d\n",m); //70
    int a=ABS1(x-y);
    int b=ABS2(x-y);
    printf("%d %d\n",a,b); //-150 10
    INTER(int,a,b);
    printf("%d %d\n",a,b); //10 -150
    INTER(int,a,b);
    printf("%d %d\n",a,b); //-150 10
    return 0;
}
```

Directiva `#define`

- ❑ folosită pentru definirea (înlocuirea) constantelor simbolice și a macro-urilor
- ❑ definirea unei **constante simbolice** este un caz special al definirii unui macro (fragment de cod care primește un nume)
`#define nume text`
- ❑ în timpul preprocesării **nume** este înlocuit cu **text**
- ❑ **text** poate să fie mai lung decât o linie, continuarea se poate face prin caracterul \ pus la sfârșitul liniei
- ❑ **text** poate să lipsească, caz în care se definește o constantă vidă

Directiva `#define`

- ❑ folosită pentru definirea (înlocuirea) constantelor simbolice și a macro-urilor
- ❑ definirea unei **constante simbolice** este un caz special al definirii unui macro (fragment de cod care primește un nume)

```
#define nume text
```

- ❑ În timpul preprocesării **nume** este înlocuit cu **text**
- ❑ Înlocuirea se continuă până în momentul în care **nume** nu mai este definit sau până la sfârșitul fișierului
 - ❑ renunțarea la definirea unei constante simbolice se poate face cu directiva `#undef nume`

Directiva #define

- ❑ definirea unui **macro**:

```
#define nume (p1, p2, ..., pn) text
```

- ❑ numele macro-ului este nume
- ❑ parametri macro-ului sunt **p1, p2, ..., pn**
- ❑ textul substituit este **text**
- ❑ parametrii formali sunt substituți de cei actuali în text
- ❑ apelul macro-ului este similar apelului unei funcții
nume(p_actual1, p_actual2, ..., p_actualn)

Directiva #define

- ❑ invocarea unui **macro** presupune înlocuirea apelului cu **textul** macro-ului respectiv
 - ❑ se generează astfel instrucțiuni la fiecare invocare și care sunt ulterior compilate
 - ❑ se recomandă astfel utilizarea doar pentru calcule simple
 - ❑ parametrul formal este înlocuit cu textul corespunzător parametrului actual, corespondența fiind pur pozitională
- ❑ timpul de procesare este mai scurt când se utilizează macro-uri (apelul funcției necesită timp suplimentar)

Compilarea condiționată

```
1 #include <stdio.h>
2
3 #define VERSION 2
4
5 int main()
6 {
7
8     #if VERSION == 1
9     {
10         printf ("versiunea 1 \n");
11         printf ("Adaugam modulele pentru versiunea 1 ... \n");
12         // continua cu includerea diverselor module pentru versiunea 1
13     }
14
15     #elif VERSION == 2
16     {
17         printf ("versiunea 2 \n");
18         printf ("Adaugam modulele pentru versiunea 2 ... \n");
19         // continua cu includerea diverselor module pentru versiunea 2
20     }
21     #elif VERSION == 3
22     {
23         printf ("versiunea 3 \n");
24         printf ("Adaugam modulele pentru versiunea 3 ... \n");
25         // continua cu includerea diverselor module pentru versiunea 3
26     }
27
28     #endif
29     return 0;
30
31 }
```

Compilarea condiționată

- ❑ facilitează dezvoltarea dar în special testarea codului
- ❑ directivele care pot fi utilizate: `#if`, `#ifdef`, `#ifndef`
- ❑ directiva `#if`:

```
#if expr  
    text  
#endif
```

```
#if expr  
    text1  
#else (#elif)  
    text2  
#endif
```

- ❑ unde `expr` este o expresie constantă care poate fi evaluată de către preprocesor, `text`, `text1`, `text2` sunt porțiuni de cod sursă
- ❑ dacă `expr` nu este zero atunci `text` respectiv `text1` sunt compilate, altfel numai `text2` este compilat și procesarea continuă după `#endif`

Compilarea condiționată

STDIO.H

```
57  *
58  *  @(#)stdio.h 8.5 (Berkeley) 4/29/95
59  */
60
61 #ifndef _STDIO_H_
62 #define _STDIO_H_
63
```

```
496 #if defined (__GNUC__) && _FORTIFY_SOURCE > 0 && !defined (__cplus)
497 /* Security checking functions.  */
498 #include <secure/_stdio.h>
499 #endif
500
501 #endif /* _STDIO_H_ */
502
```

Compilarea condiționată

- directiva **#ifdef**:

```
#ifdef nume  
    text  
#endif
```

```
#ifdef nume  
    text1  
#else  
    text2  
#endif
```

- unde **nume** este o constantă care este testată de către preprocesor dacă este definită, **text**, **text1**, **text2** sunt porțiuni de cod sursă
- dacă **nume** este definită atunci **text** respectiv **text1** sunt compilate, altfel numai **text2** este compilat și procesarea continuă după **#endif**

Compilarea condiționată

□ directiva `#ifndef`:

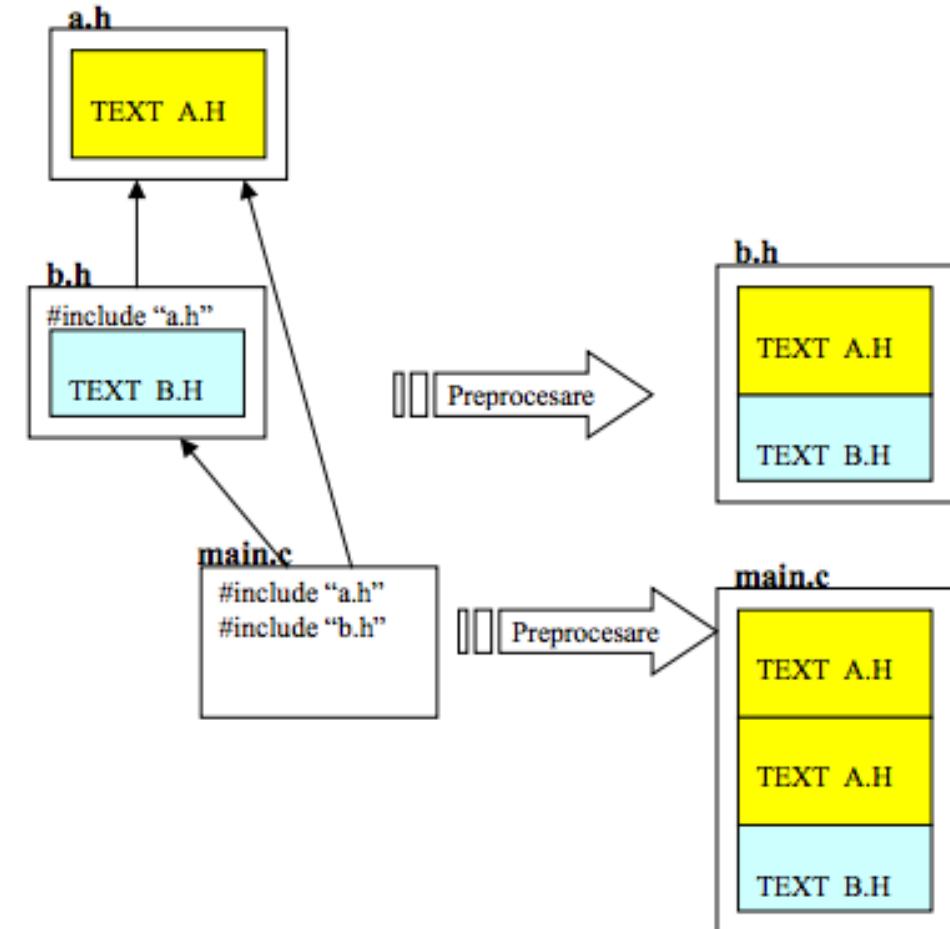
```
#ifndef nume  
    text  
#endif
```

```
#ifndef nume  
    text1  
#else  
    text2  
#endif
```

- unde `nume` este o constantă care este testată de către preprocesor dacă NU este definită, `text`, `text1`, `text2` sunt porțiuni de cod sursă
- dacă `nume` NU este definită atunci `text` respectiv `text1` sunt compilate, altfel numai `text2` este compilat și procesarea continuă după `#endif`

Compilarea condiționată

- directivele **#ifdef** și **#ifndef** sunt folosite de obicei pentru a evita incluziunea multiplă a modulelor în programarea modulară
- fișier antet “a.h”
- fișier antet “b.h”
 - include pe “a.h”
- main include “a.h” și “b.h”



Compilarea condiționată

- ❑ directivele **#ifdef** și **#ifndef** sunt folosite de obicei pentru a evita incluziunea multiplă a modulelor în programarea modulară
- ❑ fișier antet “a.h”
- ❑ fișier antet “b.h”
- ❑ la începutul fiecărui fișier *header* se practică de obicei o astfel de secvență

```
#ifndef _MODUL_A_
#define _MODUL_A_
...
#endif /* _MODUL_A_ */
```

Macro-uri predefinite

- există o serie de macro-uri predefinite care nu trebuie re/definite:

<u>DATE</u>	data compilării
<u>CDECL</u>	apelul funcției urmărește convențiile C
<u>STDC</u>	definit dacă trebuie respectate strict regulile ANSI C
<u>FILE</u>	numele complet al fișierului curent compilat
<u>FUNCTION</u>	numele funcției curente
<u>LINE</u>	numărul liniei curente

```
#include <stdio.h>
//constante simbolice
#define DEBUG
#define X -3
#define Y 5

int main()
{
#define DEBUG
    printf("Suntem in functia %s\n", __FUNCTION__); //main
#endif
#define X+Y
    double a=3.1;
#else
    double a=5.7;
#endif
    a*=2;
#define DEBUG
    printf("La linia %d valoarea lui a este %f\n", __LINE__,a); //18 6.2
#endif
    a+=10;
    printf("a este %f",a); //16.2
    return 0;
}
```

Cuprinsul cursului de azi

1. Instrucțiuni de control (break, continue, goto, return).
2. Etapele realizării unui program C.
3. Directive de preprocessare. Macrodefiniții.
4. Funcții de citire/scriere.

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:

```
exempluCitireScriere1.c  x
01
02
03
04
05
06
07
08
09
10
11
12
13
14
```

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

```
[Bogdan-Alexes-MacBook-Pro:curs5 bogdan$ ./a.out
Alegeti DA sau NU. Optiunea este : DA
| D
```

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**

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/>

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.

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 specifier

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

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	<p>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.</p> <p>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).</p> <p>For g and G specifiers: This is the maximum number of significant digits to be printed.</p> <p>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.</p> <p>If the period is specified without an explicit value for <i>precision</i>, 0 is assumed.</p>
.*	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 1:

```
exempluPrintf1.c
```

```
1 #include <stdio.h>
2
3 int main()
4 {
5     double numar;
6     numar = 10.1234;
7
8     printf("numar = %f\n", numar);
9     printf("numar = %10f\n", numar);
10    printf("numar = %012f\n", numar);
11
12    printf("%.4f\n", 123.1234567);
13    printf("%8.3d\n", 1000);
14    printf("%3.8d\n", 1000);
15    printf("%+10d\n", 1000);
16    printf("%-+10d\n", 1000);
17    printf("%10.15s\n", "Acesta este un test simplu");
18
19    return 0;
20 }
```

Modelatori de format pentru printf

exemplu 1:

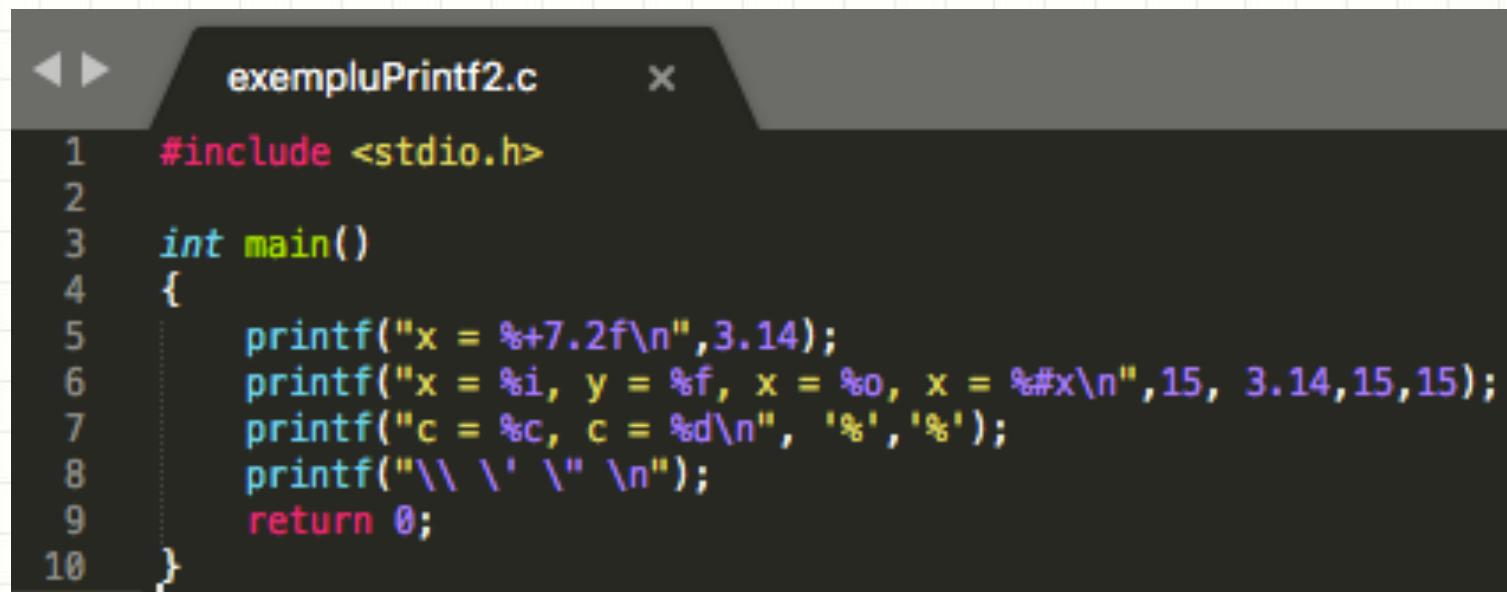
```
exempluPrintf1.c
```

```
1 #include <stdio.h>
2
3 int main()
4 {
5     double numar;
6     numar = 10.1234;
7
8     printf("numar = %f\n", numar);
9     printf("numar = %10f\n", numar);
10    printf("numar = %012f\n", numar);
11
12    printf("%.4f\n", 123.1234567);
13    printf("%8.3d\n", 1000);
14    printf("%3.8d\n", 1000);
15    printf("%+10d\n", 1000);
16    printf("%-+10d\n", 1000);
17    printf("%10.15s\n", "Acesta este un");
18
19    return 0;
20 }
```

```
Bogdan-Alexes-MacBook-Pro:curs5 bogdan$ gcc exempluPrintf1.c
Bogdan-Alexes-MacBook-Pro:curs5 bogdan$ ./a.out
numar = 10.123400
numar = 10.123400
numar = 00010.123400
123.1235
      1000
      00001000
          +1000
      +1000
Acesta este un
```

Modelatori de format pentru printf

□ exemplu 2:

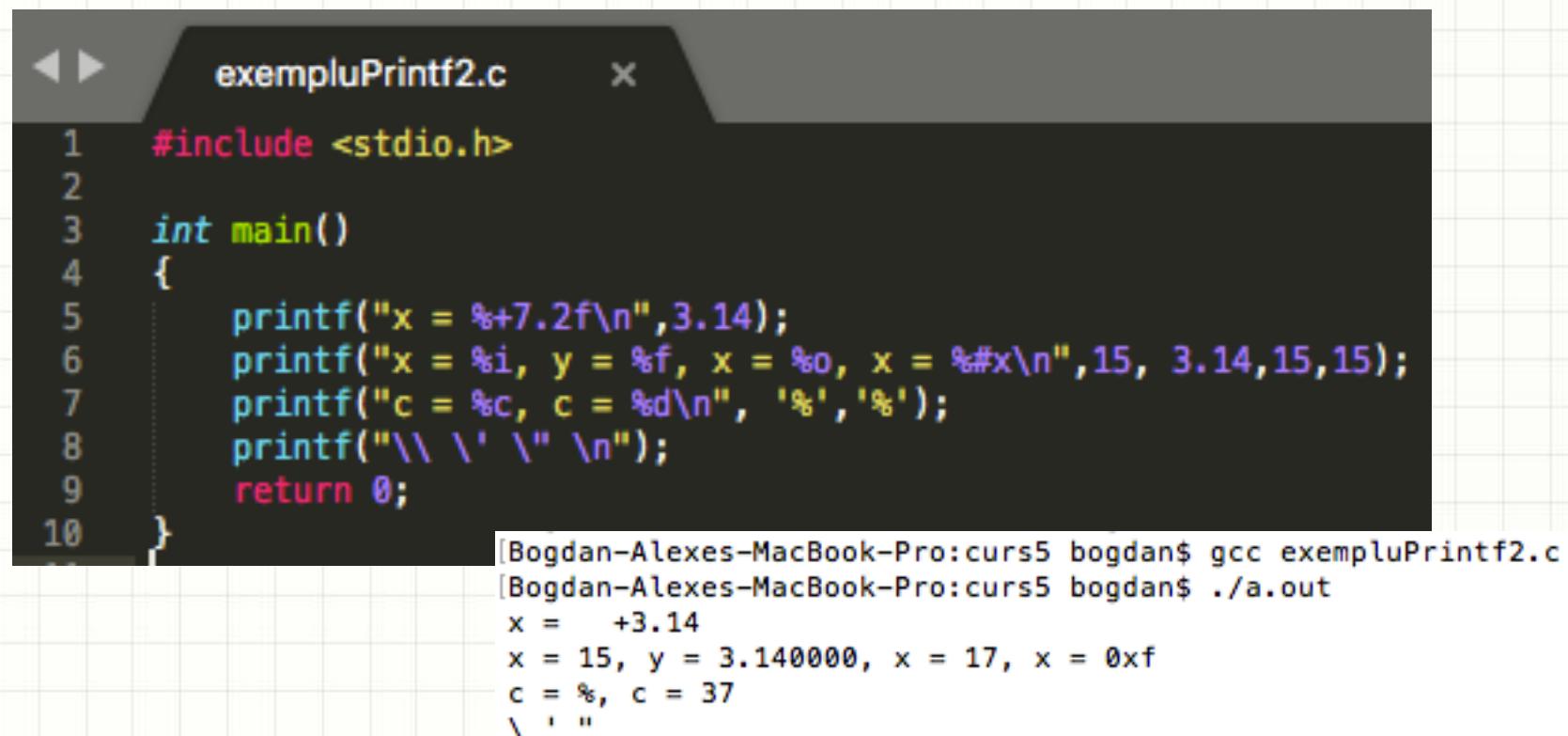


```
exempluPrintf2.c      x

1 #include <stdio.h>
2
3 int main()
4 {
5     printf("x = %+7.2f\n", 3.14);
6     printf("x = %i, y = %f, x = %o, x = %#x\n", 15, 3.14, 15, 15);
7     printf("c = %c, c = %d\n", '%', '%');
8     printf("\\ \\ \\ \\ \n");
9     return 0;
10 }
```

Modelatori de format pentru printf

exemplu 2:



```
exempluPrintf2.c      x

1 #include <stdio.h>
2
3 int main()
4 {
5     printf("x = %+7.2f\n",3.14);
6     printf("x = %i, y = %f, x = %o, x = %#x\n",15, 3.14,15,15);
7     printf("c = %c, c = %d\n", '%','%');
8     printf("\\ \\ \" \" \n");
9     return 0;
10 }
```

[Bogdan-Alexes-MacBook-Pro:curs5 bogdan\$ gcc exempluPrintf2.c
[Bogdan-Alexes-MacBook-Pro:curs5 bogdan\$./a.out
x = +3.14
x = 15, y = 3.140000, x = 17, x = 0xf
c = %, c = 37
\ \"

Functia scanf

- **prototipul functiei:**

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

unde:

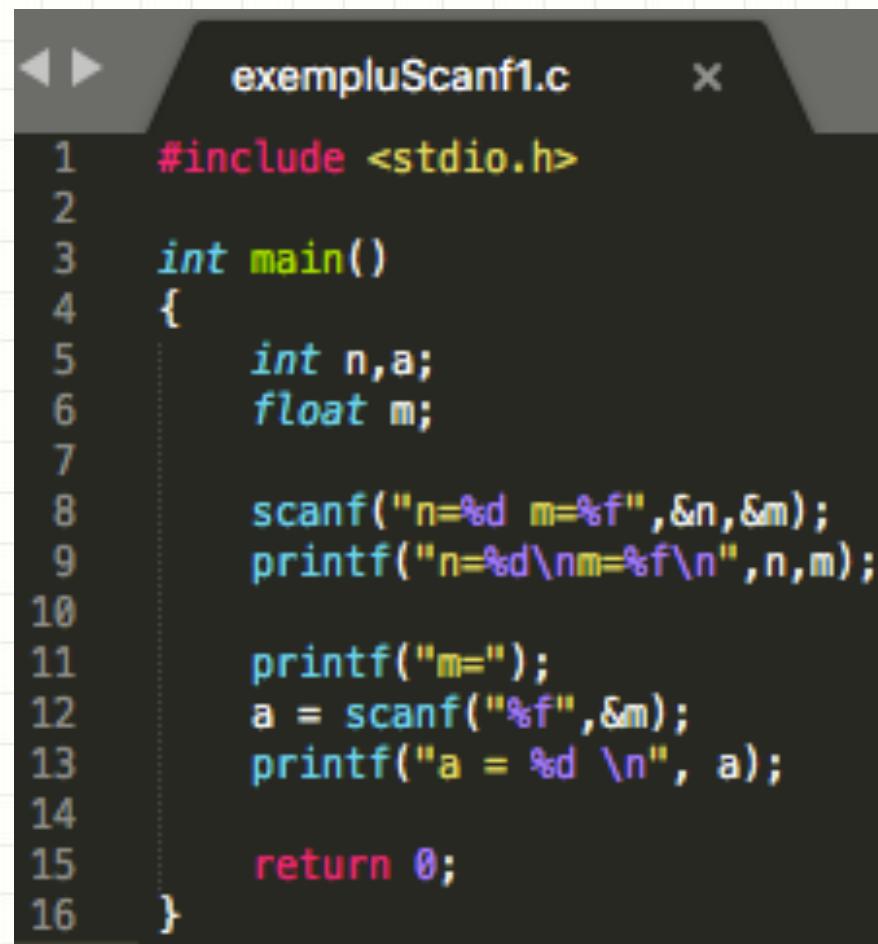
- **format** este un sir de caractere ce defineste textele si formatele datelor care se citesc de la tastatura
- **adresa1, adresa2,...** sunt adresele zonelor din memorie in care se pастreaza datele citite după ce au fost convertite din reprezentarea lor externă în reprezentare internă.
- functia **scanf** 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 1:



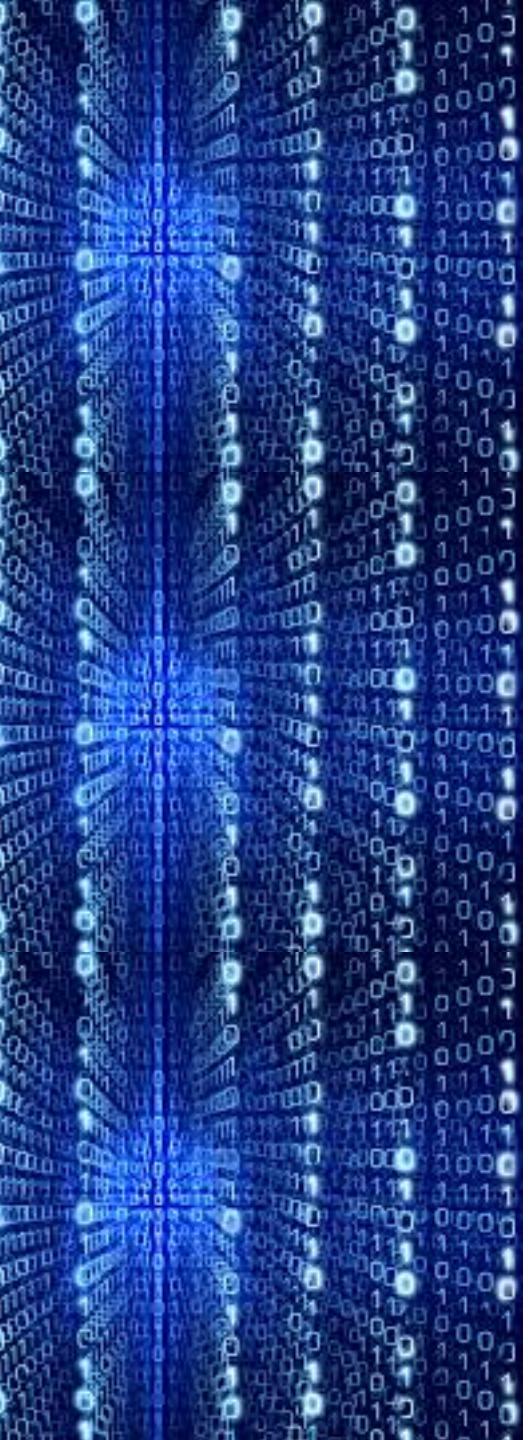
```
exempluScanf1.c      x
1 #include <stdio.h>
2
3 int main()
4 {
5     int n,a;
6     float m;
7
8     scanf("n=%d m=%f",&n,&m);
9     printf("n=%d\nm=%f\n",n,m);
10
11    printf("m=");
12    a = scanf("%f",&m);
13    printf("a = %d \n", a);
14
15    return 0;
16 }
```

Functia scanf

□ exemplu 1:

```
exempluScanf1.c      x
1 #include <stdio.h>
2
3 int main()
4 {
5     int n,a;
6     float m;
7
8     scanf("n=%d m=%f",&n,&m);
9     printf("n=%d\nm=%f\n",n,m);
10
11    printf("m=");
12    a = scanf("%f",&m);
13    printf("a = %d \n", a);
14
15    return 0;
16 }
```

```
Bogdan-Alexes-MacBook-Pro:curs5 bogdan$ ./a.out
n=25      m=2.6
n=25
m=2.600000
m=i37
a = 0
```



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,

2018-2019

Cursul 6

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, siruri 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)

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

□ Tablouri și pointeri

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

□ Siruri 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.
- Programare generică.

□ Recursivitate

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, siruri 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.

Fișiere binare

- Funcții specifice de manipulare.



□ Funcții (1)

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

□ Tablouri și pointeri

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

□ Siruri de caractere

- 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

Cuprinsul cursului de azi

1. Fișiere: noțiuni generale.
2. Fișiere text: funcții specifice de manipulare.
3. Fișiere binare: funcții specifice de manipulare.
4. Funcții

Fișiere

- **fișier = sir de octeți (colecție de date) memorat pe suport extern (magnetic, optic) și identificat printr-un nume.**

- programe sursă: .c, .cpp
- executabile
- imagini: .jpeg, .jpg, .png, .bmp
- documente: .pdf, .dvi, .eps
- audio: .mp3
- video: .avi, .mp4
- etc.

- pentru fiecare tip de fișier este necesar un program care să cunoască și să interpreze corect datele din fișier

Fișiere

- **fișier = sir de octeți (colecție de date) memorat pe suport extern (magnetic, optic) și identificat printr-un nume.**

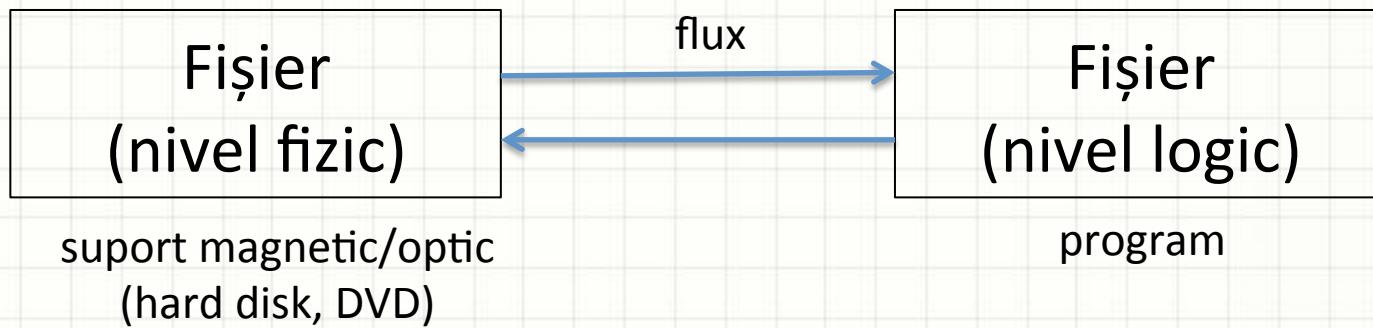
- fișierele sunt entități ale sistemului de operare.
- operațiile cu fișiere se realizează de către sistemul de operare, compilatorul de C traduce funcțiile de acces la fișiere în apeluri ale funcțiilor de sistem; alte limbiage de programare fac același lucru;

- noțiunea de fisier este mai generală
- **fișier = flux de date (stream) = transfer de informație binară (sir de octeți) de la o sursă spre o destinație:**
 - citire: flux de la tastatură (sursă) către memoria internă (destinație)
 - afișare: flux de la memoria internă (sursă) către periferice (monitor, imprimantă)

Fluxuri automate asociate unui program

- **stdin (standard input)** – flux de intrare (citire).
 - asociat implicit cu tastatura.
- **stdout (standard output)** – flux de ieșire (afișare).
 - asociat implicit cu ecranul.
- **stderr (standard error)** – flux de ieșire (afișare) pentru erori.
 - asociat implicit cu ecranul.

Fișiere



- ne referim la un fișier fizic într-un program C printr-o variabilă;
- asocierea dintre numele extern al un fișier (un sir de caractere) și o variabilă dintr-un program C se realizează la deschiderea unui fișier, printr-o funcție standard (**fopen**).

Fișiere

- **există 2 tipuri de fișiere:**
 - **fișiere text: fiecare octet este interpretat drept caracter cu codul ASCII dat de octetul respectiv**
 - organizare pe linii (\n are codul ASCII 10)
 - un fișier text în care scriem numărul întreg 123 ocupă trei octeți (codul ASCII pt 1, codul ASCII pt 2, codul ASCII pt 3)
 - **fișiere binare: octeții nu sunt organizați în nici un fel**
 - nu există noțiunea de linie
 - un fișier binar în care scriem numărul întreg 123 ocupă 4 octeți (scrierea binară a lui 123 în baza 2 pentru un int)

Fișiere

- **tipuri de fișiere:**

- **fișiere text: octeții (caractere ASCII) sunt organizați pe linii.**

Caracterele terminatorii de linii sunt:

- Unix: caracterul line feed (LF) = '\n' – cod ASCII 10
 - Mac OS vechi : caracterul carriage return (CR) = '\r' – cod ASCII 13
 - Windows: CR + LF = '\r\n'
 - un fișier text poate fi terminat printr-un caracter terminator de fișier (EOF = CTRL-Z). Acest terminator nu este obligatoriu. Sfârșitul unui fișier disc poate fi detectat și pe baza lungimii fișierului (număr de octeți), memorată pe disc.

- **fișiere binare: octeții nu sunt organizați în nici un fel (nu există noțiunea de linie)**

Fișiere

□ tipuri de fișiere:

- **fișiere text: octeții (caractere ASCII) sunt organizați pe linii.**

Caracterele terminatorii de linii sunt:

- Unix: caracterul line feed (LF) = '\n' – cod ASCII 10
- Mac OS vechi : caracterul carriage return (CR) = '\r' – cod ASCII 13
- Windows: CR + LF = '\r\n'
- un fișier text poate fi terminat printr-un caracter terminator de fișier (EOF = CTRL-Z). Acest terminator nu este obligatoriu. Sfârșitul unui fișier disc poate fi detectat și pe baza lungimii fișierului (număr de octeți), memorată pe disc.

Acesta este
un
exemplu
fișier fizic

Windows

În Windows dimensiunea fizică a unui fișier = dimensiunea logică dacă avem o singură linie

Acesta este\r\nun\r\n\r\nexemplu
fișier logic

Lucrul cu fișiere

- În biblioteca stdio.h este definită o structură **FILE**. Această structură conține (în membri ei) informații referitoare la un fișier: nume, adresa, adresa bufferului intern în care se procesează (citire/scriere) octetii din fișier, indicator de sfârșit de fișier, indicator de poziție în fișier, etc.
- Lucrul cu fișiere presupune declararea unui pointer la structura **FILE** în vederea realizării legăturii dintre nivelul logic (variabila fișier) și nivelul fizic (numele extern al fișierului) :

FILE * f;

- etapele pentru lucrul cu fișiere:
 - deschiderea unui fișier – funcția **fopen** (legătura nivel logic-fizic);
 - prelucrarea fișierului (citiri/scrieri – diferă pentru fișiere text și cele binare) ;
 - închiderea fișierului – funcția **fclose**;

Lucrul cu fișiere

- **deschiderea unui fișier = stabilirea unui flux către acel fișier.** Se realizează folosind funcția fopen:
- **sintaxa**

FILE *fopen(char *nume_fisier, char *mod_deschidere)

unde

- nume_fisier = numele fisierului
- mod_deschidere = sir de caracter (1-3 caractere) ce precizează tipul de acces la fișier:
 - citire "r", scriere "w", adăugare la sfârșitul fisierului "a";
 - "+" permite scrierea și citirea "r+", "w+", "a+";
 - t (implicit) sau b: specifică tipul de fisier (text sau binar).
- funcția **fopen** întoarce un pointer la o structura **FILE** sau în caz de eroare (fișierul nu se poate deschide) întoarce NULL.

Lucrul cu fișiere

- **deschiderea unui fișier = stabilirea unui flux către acel fișier.** Se realizează folosind funcția fopen:
- **sintaxa**

FILE *fopen(char *nume_fisier, char *mod_deschidere)

unde mod_deschidere = sir de caracter (1-3 caractere) ce precizează tipul de acces la fișier:

Mod	Semnificație
r	Deschide un fișier tip text pentru a fi citit
w	Creează un fișier tip text pentru a fi scris
a	Adaugă într-un fișier tip text
rb	Deschide un fișier de tip binar pentru a fi citit
wb	Creează un fișier de tip binar pentru a fi scris
ab	Adaugă într-un fișier de tip binar
r+	Deschide un fișier tip text pentru a fi citit/scris
w+	Creează un fișier tip text pentru a fi citit/scris
a+	Adaugă în sau creează un fișier tip text pentru a fi citit/scris
r+b	Deschide un text în binar pentru a fi citit/scris
w+b	Creează un fișier de tip binar pentru a fi citit/scris
a+b	Adaugă sau creează un fișier de tip binar pentru a fi citit/scris

Lucrul cu fișiere

- **Închiderea unui fișier = Închiderea unui flux către acel fișier.** Se realizează folosind funcția fclose:
- **sintaxa**

int fclose(FILE *f)

unde f = pointer la structura de tip FILE care realizează legătura cu fișierul pe care vreau să-l închid

- funcția **fclose** întoarce valoarea 0 dacă închiderea s-a efectuat cu succes și EOF în caz de eroare. Toate fișierele în care s-a scris trebuie să fie închise. Dacă se realizează doar citirea dintr-un fișier, acesta nu trebuie neapărat închis.
- **tastatura și imprimanta** sunt considerate **fișiere text**. Ele nu trebuie să fie deschise și închise.

Lucrul cu fișiere

```
exempluFopen.c x

1 #include<stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     FILE *f;
7     char *numeFisier = "exempluFopen.c";
8     f = fopen(numeFisier,"r");
9     if(f==NULL)
10    {
11        printf("Eroare la deschiderea fisierului");
12        return 0;
13    }
14    else
15        printf("Am deschis fisierul %s in mod text \n",numeFisier);
16    fclose(f);
17    printf("Am inchis fisierul %s deschis in mod text \n",numeFisier);
18
19    f = fopen(numeFisier,"rb");
20    if(f==NULL)
21    {
22        printf("Eroare la deschiderea fisierului");
23        return 0;
24    }
25    else
26        printf("Am deschis fisierul %s in mod binar\n",numeFisier);
27    fclose(f);
28    printf("Am inchis fisierul %s deschis in mod binar\n",numeFisier);
29
30
31    return 0;
32 }
```

Lucrul cu fișiere

```
exempluFopen.c
```

```
1 #include<stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     FILE *f;
7     char *numeFisier = "exempluFopen.c";
8     f = fopen(numeFisier,"r");
9     if(f==NULL)
10    {
11        printf("Eroare la deschiderea fisierului");
12        return 0;
13    }
14    else
15        printf("Am deschis fisierul %s in mod text \n",numeFisier);
16    fclose(f);
17    printf("Am inchis fisierul %s deschis in mod text \n",numeFisier);
18
19    f = fopen(numeFisier,"rb");
20    if(f==NULL)
21    {
22        printf("Eroare la deschiderea fisierului");
23        return 0;
24    }
25    else
26        printf("Am deschis fisierul %s in mod binar \n",numeFisier);
27    fclose(f);
28    printf("Am inchis fisierul %s deschis in mod binar \n",numeFisier);
29
30
31    return 0;
32 }
```

```
Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ gcc exempluFopen.c
Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ ./a.out
Am deschis fisierul exempluFopen.c in mod text
Am inchis fisierul exempluFopen.c deschis in mod text
Am deschis fisierul exempluFopen.c in mod binar
Am inchis fisierul exempluFopen.c deschis in mod binar
```

Pentru Windows, atentie la folosirea lui "\\" (folosit la sevențe escape) pentru a da calea fișierului: f = fopen("C:\\work\\test.txt"); // in Windows

Functii pentru lucrul cu fisierele

- se folosesc pentru ambele tipuri de fisiere (text și binare)
- **FILE *fopen(char *nume_fisier, char *mod)**
 - deschide fisierul cu numele *nume_fisier* pentru acces de tip *mod*
- **int fclose(FILE * f);**
 - închide fisierul asociat cu variabile f și eliberează bufferul;
- **int feof(FILE *f);**
 - returnează 0 dacă nu s-a detectat sfârșit de fisier (EOF) la ultima operație de citire, altfel o valoare nenulă pentru EOF;
- **FILE * freopen(char *nume_fisier, char* mod, FILE* f)**
 - se închide fisierul f, se deschide fisierul nume_fisier în modul mod și se asociază cu f (de obicei f este stdin sau stdout);
- **int fflush(FILE* f)**
 - golește bufferul asociat unui fisier; pentru fisierele deschise pentru scriere are ca efect scrierea în fisier a datelor din buffer care încă nu au fost puse în fisier

Functii pentru lucrul cu fisierele

- **funcții de poziționare într-un fișier**
- În C ne putem poziționa pe un anumit octet din fișier.
Functiile care permit poziționarea (cele mai importante) sunt:
- **long ftell(FILE *f)**
 - Întoarce numărul octetului curent față de începutul fișierului;
 - (dimensiunea maximă a unui fișier în C este de $2^{31}-1$ octeți ~ 2GB)
- **int fseek(FILE *f, int nr_octeti, int origine)**
 - mută pointerul de fișier f pe octetul numărul nr_octeti in raport cu origine
 - origine – 3 valori posibile:
 - SEEK_SET (= 0) - început de fișier
 - SEEK_CUR (=1) – poziția curentă
 - SEEK_END (=2) – sfârșit de fișier

Functii de pozitionare intr-un fisier

Exemplu: aflarea dimensiunii unui fisier

exempluCalculeazaDimensiune.c

```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main()
5 {
6
7     FILE *f;
8     int numarLinii = 0;
9     f = fopen("exempluCalculeazaDimensiune.c","r");
10
11    if (f == NULL)
12    {
13        printf("Eroare la deschiderea fisierului");
14        exit(0);
15    }
16
17    fseek(f,0,SEEK_END);
18    long int nbBytes = ftell(f);
19    printf("Fisierul are %ld octeti\n",nbBytes);
20
21    fclose(f);
22    return 0;
23
24 }
```

```
Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ ./a.out
Fisierul are 363 octeti
```

exempluCalculeazaDimensiune.c

```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main()
5 {
6
7     FILE *f;
8     int numarLinii = 0;
9     f = fopen("exempluCalculeazaDimensiune.c","rb");
10
11    if (f == NULL)
12    {
13        printf("Eroare la deschiderea fisierului");
14        exit(0);
15    }
16
17    fseek(f,0,SEEK_END);
18    long int nbBytes = ftell(f);
19    printf("Fisierul are %ld octeti\n",nbBytes);
20
21    fclose(f);
22    return 0;
23
24 }
```

```
Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ ./a.out
Fisierul are 364 octeti
```

Functii de pozitionare intr-un fisier

Exemplu: aflarea dimensiunii unui fisier

```
exempluCalculeazaDimensiune.c  x

1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main()
5 {
6
7     FILE *f;
8     int numarLinii = 0;
9     f = fopen("exempluCalculeazaDimensiune.c","r");
10
11    if (f == NULL)
12    {
13        printf("Eroare la deschiderea fisierului");
14        exit(0);
15    }
16
17    fseek(f,0,SEEK_END);
18    long int nbBytes = ftell(f);
19    printf("Fisierul are %ld octeti\n",nbBytes);
20
21    fclose(f);
22    return 0;
23
24 }
```

Bogdan-Alexes-MacBook-Pro:curs6 bogdan\$./a.out
Fisierul are 363 octeti

```
exempluCalculeazaDimensiune.c  x

1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main()
5 {
6
7     FILE *f;
8     int numarLinii = 0;
9     f = fopen("exempluCalculeazaDimensiune.c","rb");
10
11    if (f == NULL)
12    {
13        printf("Eroare la deschiderea fisierului");
14        exit(0);
15    }
16
17    fseek(f,0,SEEK_END);
18    long int nbBytes = ftell(f);
19    printf("Fisierul are %ld octeti\n",nbBytes);
20
21    fclose(f);
22    return 0;
23
24 }
```

Bogdan-Alexes-MacBook-Pro:curs6 bogdan\$./a.out
Fisierul are 364 octeti

Functii pentru lucrul cu fisierele

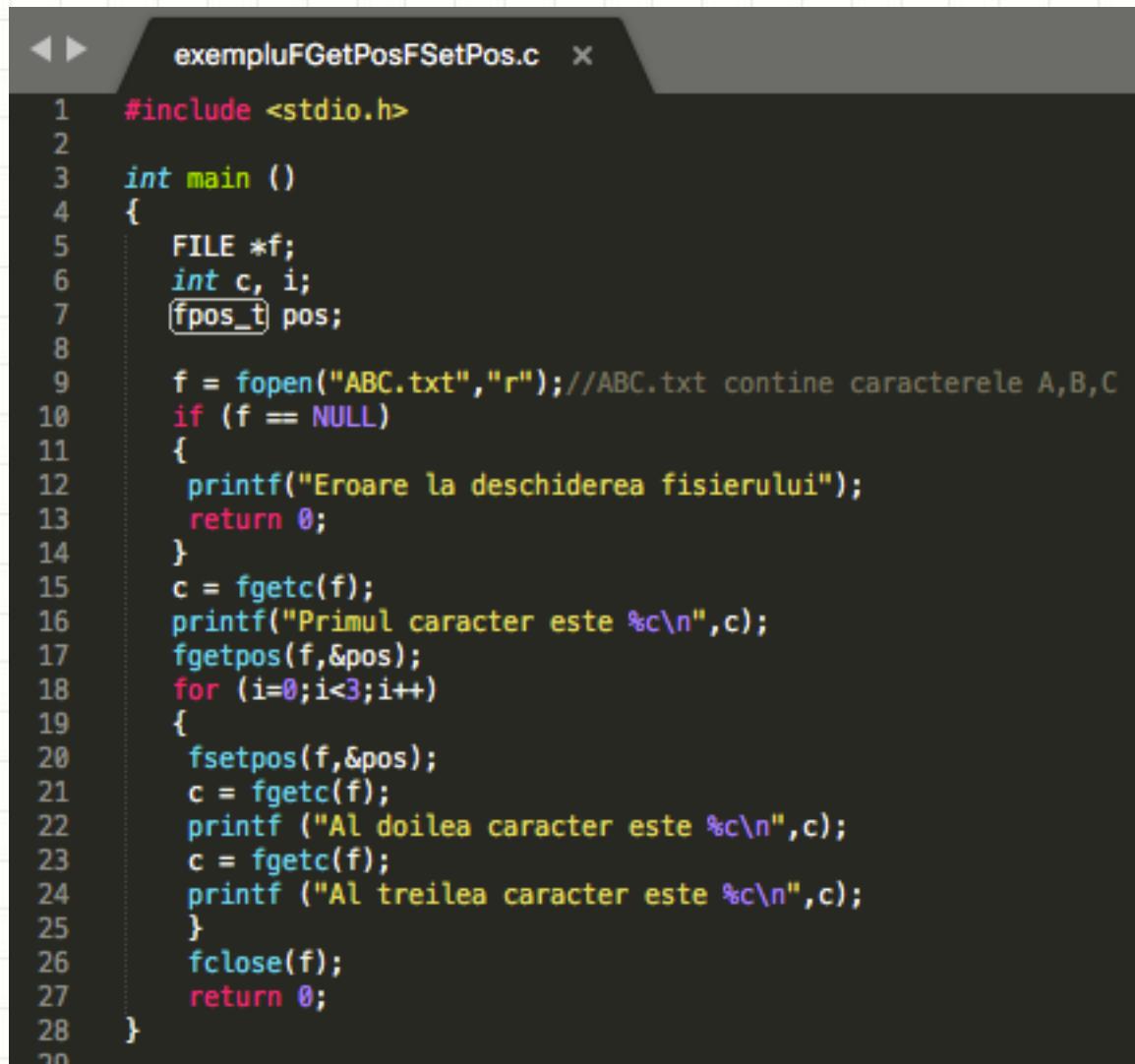
- **alte funcții de poziționare într-un fișier**

- **int fgetpos(FILE *f, const fpos_t *ptr)**
 - memorează poziția curentă în variabila ptr în cadrul fișierului asociat cu f (ptr va fi folosit ulterior cu funcția fsetpos);

- **int fsetpos (FILE *f, const fpos_t *ptr)**
 - setează poziția curentă în fișierul asociat cu f la valoarea ptr, obținută anterior prin funcția fgetpos

Functii de pozitionare într-un fișier

Exemplu:



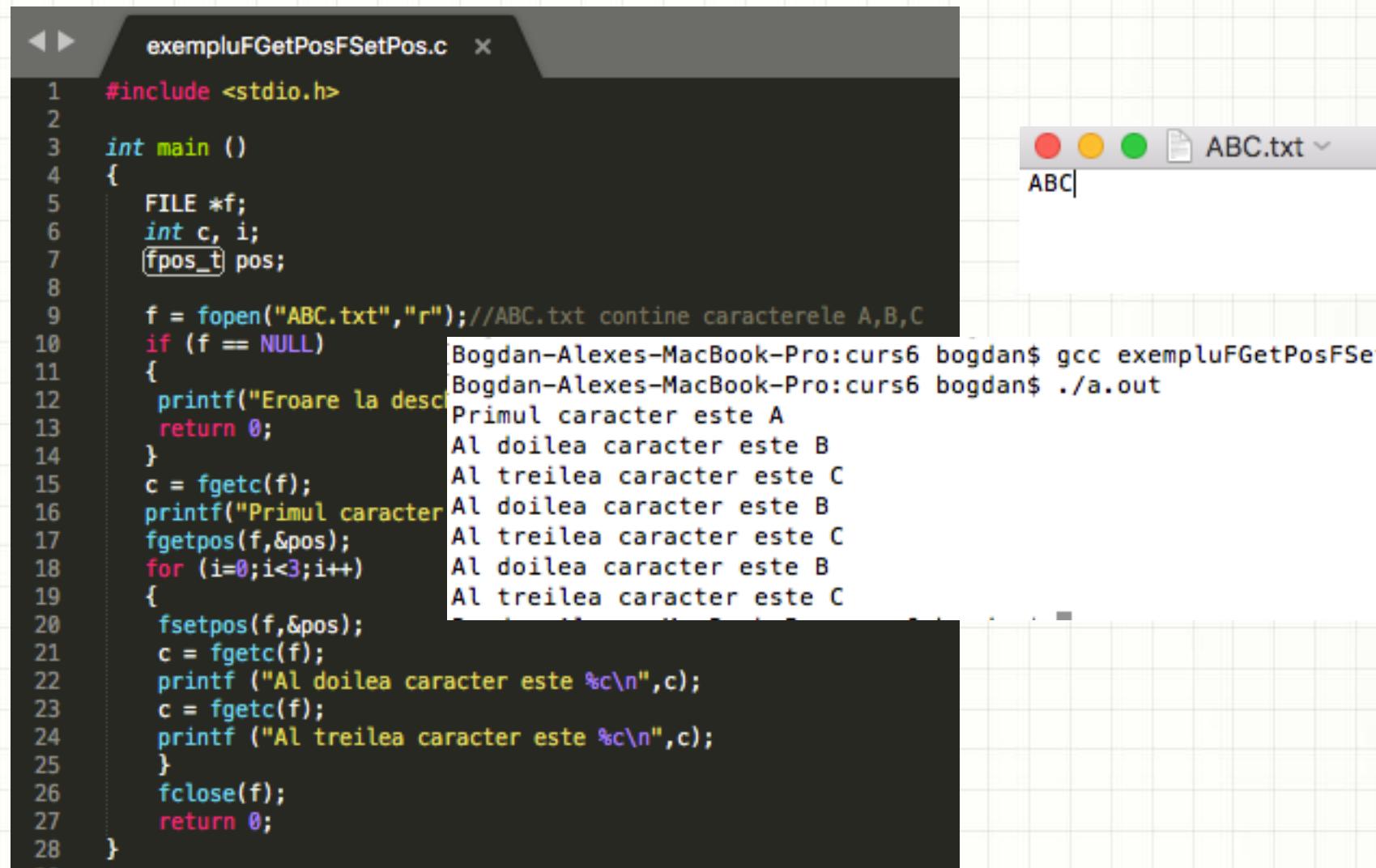
The image shows a screenshot of a code editor with a dark theme. The title bar of the editor window says "exempluFGetPosFSetPos.c". The code itself is a C program that demonstrates file positioning. It includes headers, declares variables, opens a file, reads its first three characters, prints them, and then sets the position back to the start to read the second character again.

```
#include <stdio.h>
int main ()
{
    FILE *f;
    int c, i;
    fpos_t pos;

    f = fopen("ABC.txt","r");//ABC.txt contine caracterele A,B,C
    if (f == NULL)
    {
        printf("Eroare la deschiderea fisierului");
        return 0;
    }
    c = fgetc(f);
    printf("Primul caracter este %c\n",c);
    fgetpos(f,&pos);
    for (i=0;i<3;i++)
    {
        fsetpos(f,&pos);
        c = fgetc(f);
        printf ("Al doilea caracter este %c\n",c);
        c = fgetc(f);
        printf ("Al treilea caracter este %c\n",c);
    }
    fclose(f);
    return 0;
}
```

Functii de pozitionare într-un fișier

Exemplu:



```
00 exempluFGetPosFSetPos.c  X
01
02 #include <stdio.h>
03
04 int main ()
05 {
06     FILE *f;
07     int c, i;
08     fpos_t pos;
09
10    f = fopen("ABC.txt","r");//ABC.txt contine caracterele A,B,C
11    if (f == NULL)
12    {
13        printf("Eroare la deschidere\n");
14        return 0;
15    }
16    c = fgetc(f);
17    printf("Primul caracter este %c\n",c);
18    fgetpos(f,&pos);
19    for (i=0;i<3;i++)
20    {
21        fsetpos(f,&pos);
22        c = fgetc(f);
23        printf ("Al doilea caracter este %c\n",c);
24        c = fgetc(f);
25        printf ("Al treilea caracter este %c\n",c);
26    }
27    fclose(f);
28    return 0;
29 }
```

Bogdan-Alexes-MacBook-Pro:curs6 bogdan\$ gcc exempluFGetPosFSetPos.c
Bogdan-Alexes-MacBook-Pro:curs6 bogdan\$./a.out
Primul caracter este A
Al doilea caracter este B
Al treilea caracter este C
Al doilea caracter este B
Al treilea caracter este C
Al doilea caracter este B
Al treilea caracter este C

Alte funcții pentru lucrul cu fișierele

- ❑ **void rewind (FILE *f)**
 - ❑ reposiționarea pointerului asociat fișierului la începutul său.
- ❑ **int remove(char * nume_fisier);**
 - ❑ șterge fișierul cu numele = nume_fisier. Întoarce 0 în caz de succes, 1 în caz de eroare;
- ❑ **int rename(char *nume_vechi,char *nume_nou);**
 - ❑ redenumește fișierul cu numele = nume_vechi cu nume_nou. Întoarce 0 în caz de succes, 1 în caz de eroare;
- ❑ **char *tmpnam(char* nume_fisier)**
 - ❑ furnizează un nume de fisier pe care îl pune în nume_fisier care nu există în directorul curent

Cuprinsul cursului de azi

1. Fișiere: noțiuni generale.
2. Fișiere text: funcții specifice de manipulare.
3. Fișiere binare: funcții specifice de manipulare.
4. Funcții

Fișiere text

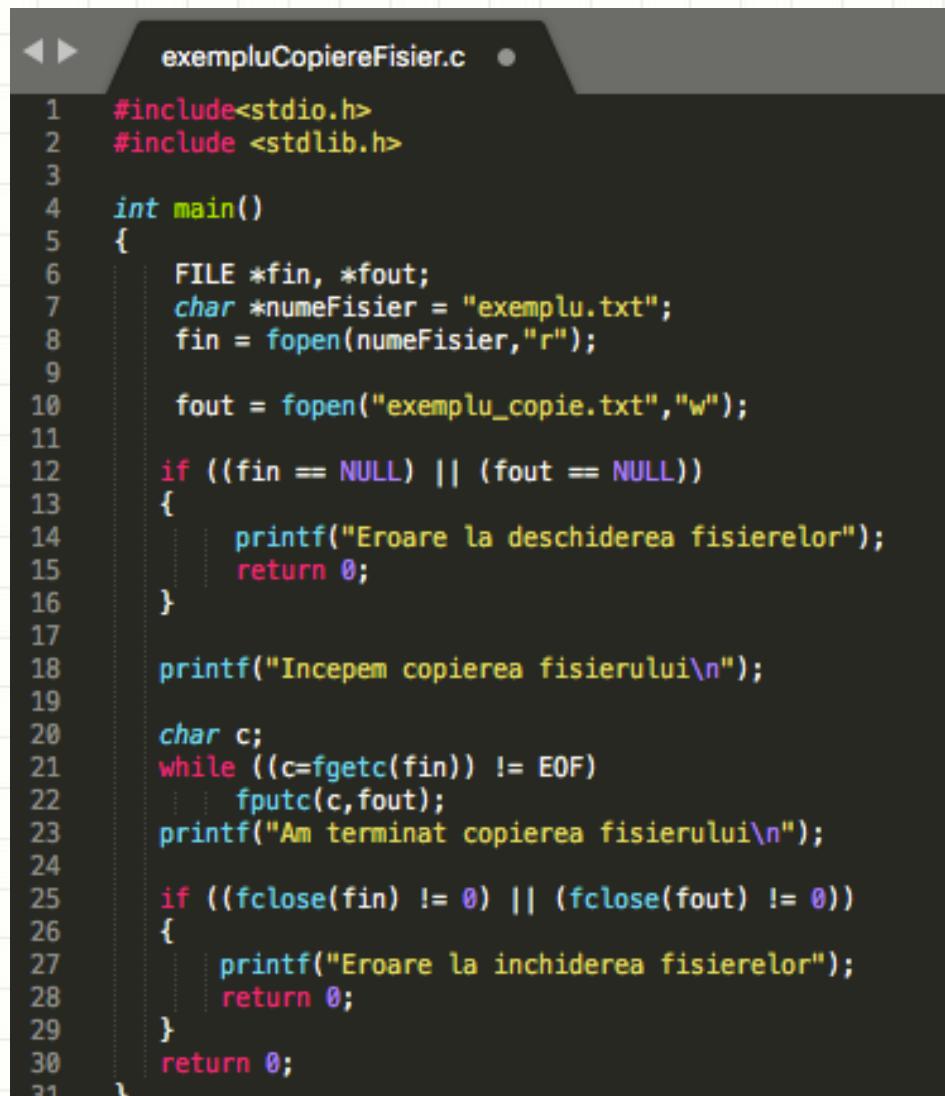
- accesul la fișierele text se poate face la nivel de **șir de caractere** (linie) sau la nivel de **caracter** (octet).
- un fișier text se accesează ca o succesiune de linii de text de lungime variabilă (încheiate cu un terminator de linie : '\n') utilizând un set dedicat de funcții din biblioteca standard.
- funcțiile de citire sau de scriere cu format din/în fișiere text realizează conversia automată din:
 - format extern (șir de caractere) în format intern (binar) - la citire
 - format intern (binar) în format extern (șir de caractere), la scriere pentru numere întregi sau reale.

Functii de citire/scriere la nivel de caracter

- **int fgetc(FILE *f)** întoarce codul ASCII al caracterului citit din fișierul f.
 - dacă s-a ajuns la finalul fișierului sau a avut loc o eroare la citire întoarce EOF (= -1).
- **int fputc(int c, FILE *f)** scrie caracterul cu codul ASCII c în fișierul f.
 - întoarce EOF (= -1) în caz de eroare sau codul ASCII al caracterului scris în caz de succes.

Functii de citire/scriere la nivel de caracter

□ Exemplu: copierea unui fisier text



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

int main()
{
    FILE *fin, *fout;
    char *numeFisier = "exemplu.txt";
    fin = fopen(numeFisier,"r");

    fout = fopen("exemplu_copie.txt","w");

    if ((fin == NULL) || (fout == NULL))
    {
        printf("Eroare la deschiderea fisierelor");
        return 0;
    }

    printf("Incepem copierea fisierului\n");

    char c;
    while ((c=fgetc(fin)) != EOF)
        fputc(c,fout);
    printf("Am terminat copierea fisierului\n");

    if ((fclose(fin) != 0) || (fclose(fout) != 0))
    {
        printf("Eroare la inchiderea fisierelor");
        return 0;
    }
    return 0;
}
```

Functii de citire/scriere la nivel de caracter

Exemplu: copierea unui fisier text

```
exempluCopiereFisier.c

1 #include<stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     FILE *fin, *fout;
7     char *numeFisier = "exemplu.txt";
8     fin = fopen(numeFisier,"r");
9
10    fout = fopen("exemplu_copie.txt","w");
11
12    if ((fin == NULL) || (fout == NULL))
13    {
14        printf("Eroare la deschiderea fisierelor");
15        return 0;
16    }
17
18    printf("Incepem copierea fisierului\n");
19
20    ...
```

```
[Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ gcc exempluCopiereFisier.c -o copiazaFisiere
```

```
[Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ ./copiazaFisiere
```

```
Incepem copierea fisierului
```

```
Am terminat copierea fisierului
```

```
25 if ((fclose(fin) != 0) || (fclose(fout) != 0))
```

```
copiazaFisiere
```

```
Today, 23:26
```

```
9 KB
```

```
Unix executable
```

```
exemplu_copie.txt
```

```
Today, 23:27
```

```
5 bytes
```

```
Plain Text
```

```
exemplu.txt
```

```
Today, 23:26
```

```
5 bytes
```

```
Plain Text
```

Functii de citire scriere la nivel de linie

- `char* fgets(char *sir, int m, FILE *f)`
 - citește maxim $m-1$ caractere sau până la '\n' și pune șirul de caractere în sir (adaugă la sfârșit '\0').
 - returnează adresa șirului citit.
 - dacă apare vreo eroare întoarce NULL.

- `int fputs(char *sir, FILE *f)`
 - scrie șirul sir în fișierul f, fără a pune '\n' la sfârșit.
 - întoarce numarul de caractere scrise, sau EOF in caz de eroare.

Functii de citire scriere la nivel de linie

- Exemplu: aflarea numărului de linii al unui fișier

```
exempluCitireLinii.c  x

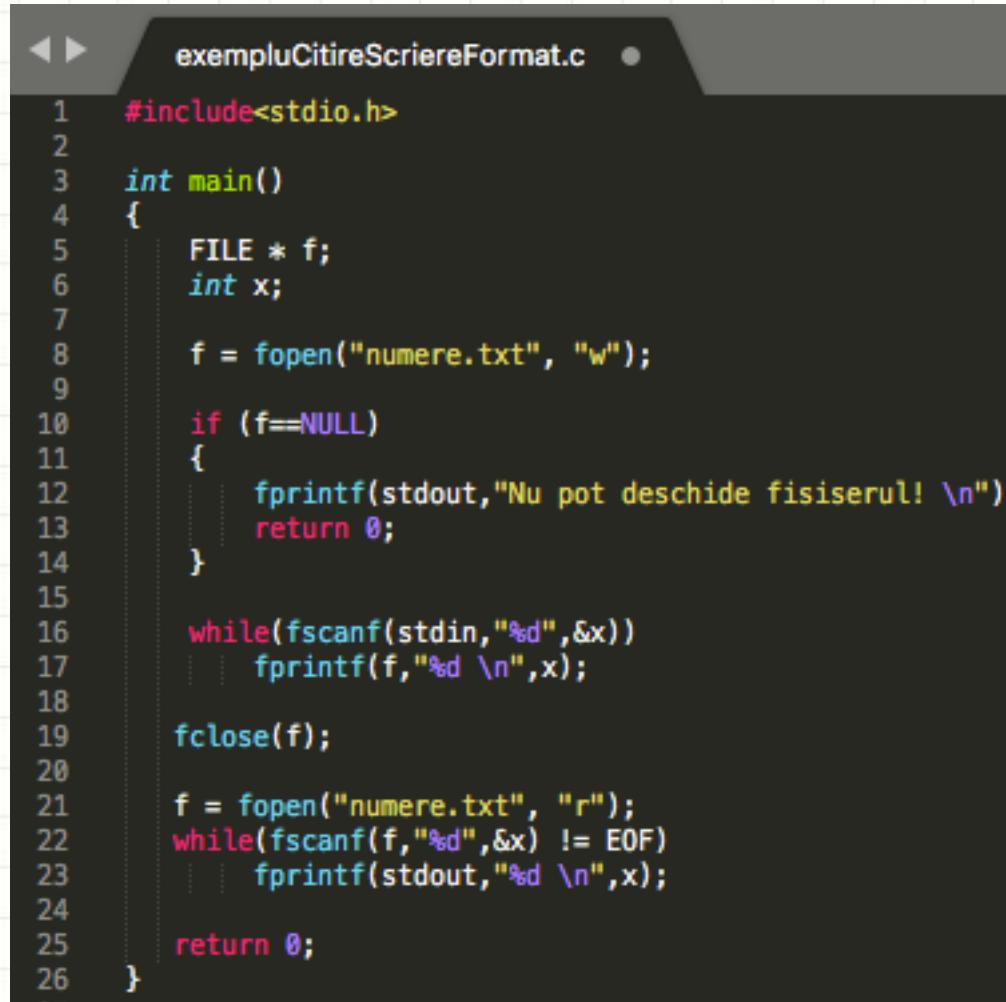
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main()
5 {
6
7     FILE *f;
8     char linie[1000];
9     int numarLinii = 0;
10    f = fopen("test.txt","r");
11
12    if (f == NULL)
13    {
14        printf("Eroare la deschiderea fisierului");
15        exit(0);
16    }
17
18    while (fgets(linie,1000,f) != NULL)
19    {
20        numarLinii++;
21    }
22
23    printf("Fisierul are %d linii\n",numarLinii);
24
25    fclose(f);
26    return 0;
27 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ gcc exempluCitireLinii.c
[Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ ./a.out
Fisierul are 242 linii
```

Functii de citire scriere cu format

- `int fscanf(FILE *f, char *format, ...)`
 - citește din fisierul f folosind un format (analog cu scanf)
- `int fprintf(FILE *f, char *format, ...)`
 - scrie în fișierul f folosind un format (analog cu printf)

Functii de citire scriere cu format



The image shows a code editor window with the title "exempluCitireScriereFormat.c". The code is written in C and demonstrates how to read and write integers from/to a file named "numere.txt" using formatted I/O.

```
#include<stdio.h>
int main()
{
    FILE * f;
    int x;
    f = fopen("numere.txt", "w");
    if (f==NULL)
    {
        fprintf(stderr,"Nu pot deschide fisierul! \n");
        return 0;
    }
    while(fscanf(stdin,"%d",&x))
        fprintf(f,"%d \n",x);
    fclose(f);
    f = fopen("numere.txt", "r");
    while(fscanf(f,"%d",&x) != EOF)
        fprintf(stdout,"%d \n",x);
    return 0;
}
```

Functii de citire scriere cu format

```
exempluCitireScriereFormat.c
```

```
1 #include<stdio.h>
2
3 int main()
4 {
5     FILE * f;
6     int x;
7
8     f = fopen("numere.txt", "w");
9
10    if (f==NULL)
11    {
12        fprintf(stdout,"Nu pot deschide fisierul! \n");
13        return 0;
14    }
15
16    while(fscanf(stdin,"%d",&x))
17    {
18        fprintf(f,"%d \n",x);
19    }
20
21
22 Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ gcc exempluCitireScriereFormat.c
23 Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ ./a.out
24
25 23
26 -1000
27 0
28 11
29 y
30 23
31 -1000
32 0
33 11
```



The terminal window shows the command line and the output of the program. The file 'numere.txt' contains the numbers 23, -1000, 0, 11, followed by a 'y'. The program reads these numbers from standard input and writes them to 'numere.txt'.

Lucrul cu fișiere

- **detectarea sfârșitului de fișier.** Se poate realiza și folosind funcția feof(find end of file) :
- **sintaxa**

int feof(FILE *f)

unde f = pointer la fișierul pe care îl prelucrez.

- funcția feof returnează 0 dacă nu s-a ajuns la sfârșitul fișierului la ultima operație de citire sau o valoare nenulă dacă s-a ajuns la sfârșitul fișierului.

ATENTIE:

```
while (!feof(f))  
    citeste x din f;  
    scrie x
```

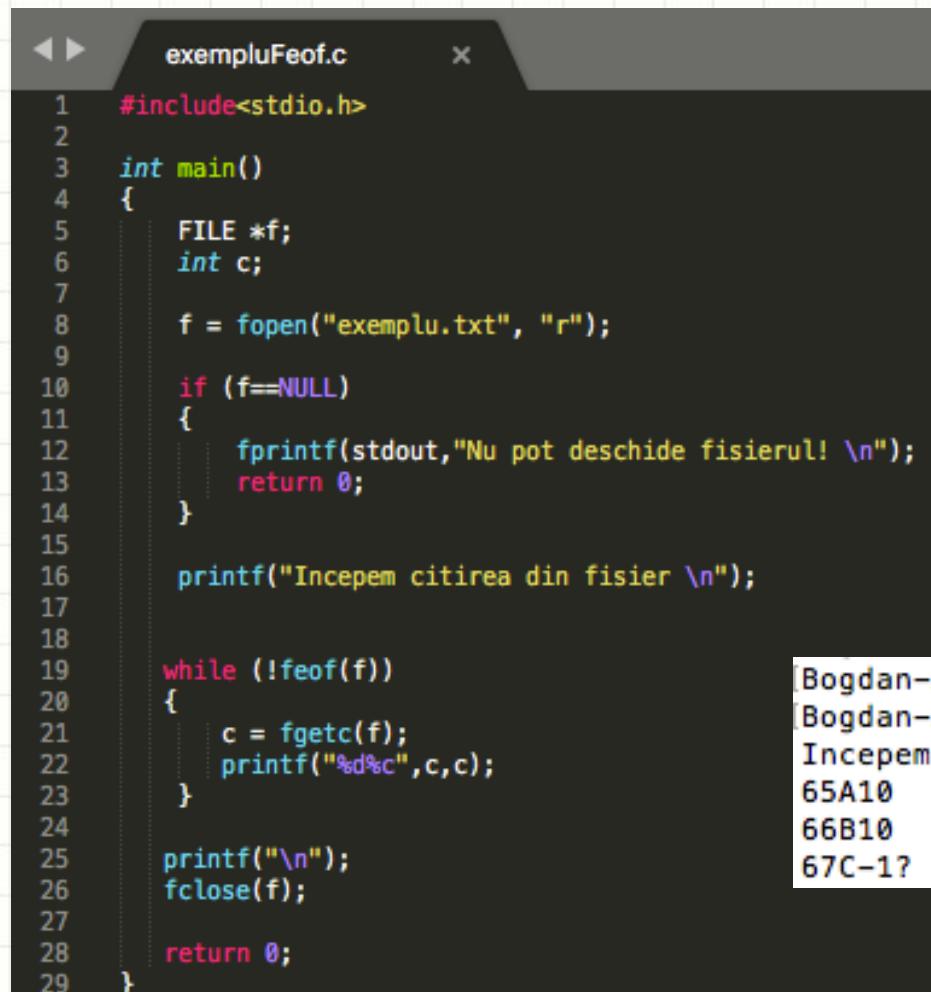
se scrie un x (= -1) în plus

varianta corecta

```
while (1)  
    citeste x din f;  
    if (feof(f)) break;  
    scrie x
```

Lucrul cu fișiere

- detectarea sfârșitului de fișier. Se poate realiza și folosind funcția feof(find end of file) :



```
#include<stdio.h>
int main()
{
    FILE *f;
    int c;

    f = fopen("exemplu.txt", "r");

    if (f==NULL)
    {
        fprintf(stdout,"Nu pot deschide fisierul! \n");
        return 0;
    }

    printf("Incepem citirea din fisier \n");

    while (!feof(f))
    {
        c = fgetc(f);
        printf("%d%c",c,c);
    }

    printf("\n");
    fclose(f);

    return 0;
}
```



```
Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ gcc exempluFeof.c
Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ ./a.out
Incepem citirea din fisier
65A10
66B10
67C-1?
```

Lucrul cu fișiere

- detectarea sfârșitului de fișier. Se poate realiza și folosind funcția feof(find end of file) :

```
exempluFeof.c
1 #include<stdio.h>
2
3 int main()
4 {
5     FILE *f;
6     int c;
7
8     f = fopen("exemplu.txt", "r");
9
10    if (f==NULL)
11    {
12        fprintf(stdout,"Nu pot deschide fisierul! \n");
13        return 0;
14    }
15
16    printf("Incepem citirea din fisier \n");
17
18
19    while (1)
20    {
21        c = fgetc(f);
22        if(feof(f))
23            break;
24        printf("%d%c",c,c);
25    }
26
27    printf("\n");
28    fclose(f);
29
30
31 }
```



```
Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ gcc exempluFeof.c
Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ ./a.out
Incepem citirea din fisier
65A10
66B10
67C
```

Cuprinsul cursului de azi

1. Fișiere: noțiuni generale.
2. Fișiere text: funcții specifice de manipulare.
3. Fișiere binare: funcții specifice de manipulare.
4. Funcții

Fișiere binare

- **fișier binar = sir de octeți neformatat pe linii care este stocat pe suport magnetic/optic.**
- un fișier binar este format în general din articole de lungime fixă, fără separatori între articole. Un articol poate conține:
 - un singur octet
 - un număr binar (pe 2, 4 sau 8 octeți)
 - structură cu date de diferite tipuri
- un fișier binar se accesează ca o succesiune de octeți, cărora funcțiile de citire și scriere din fișier nu le dau nici o interpretare.
- un fișier text se accesează ca o succesiune de linii de text de lungime variabilă (încheiate cu un terminator de linie : '\n') utilizând un set dedicat de funcții din biblioteca standard.

Fisiere binare

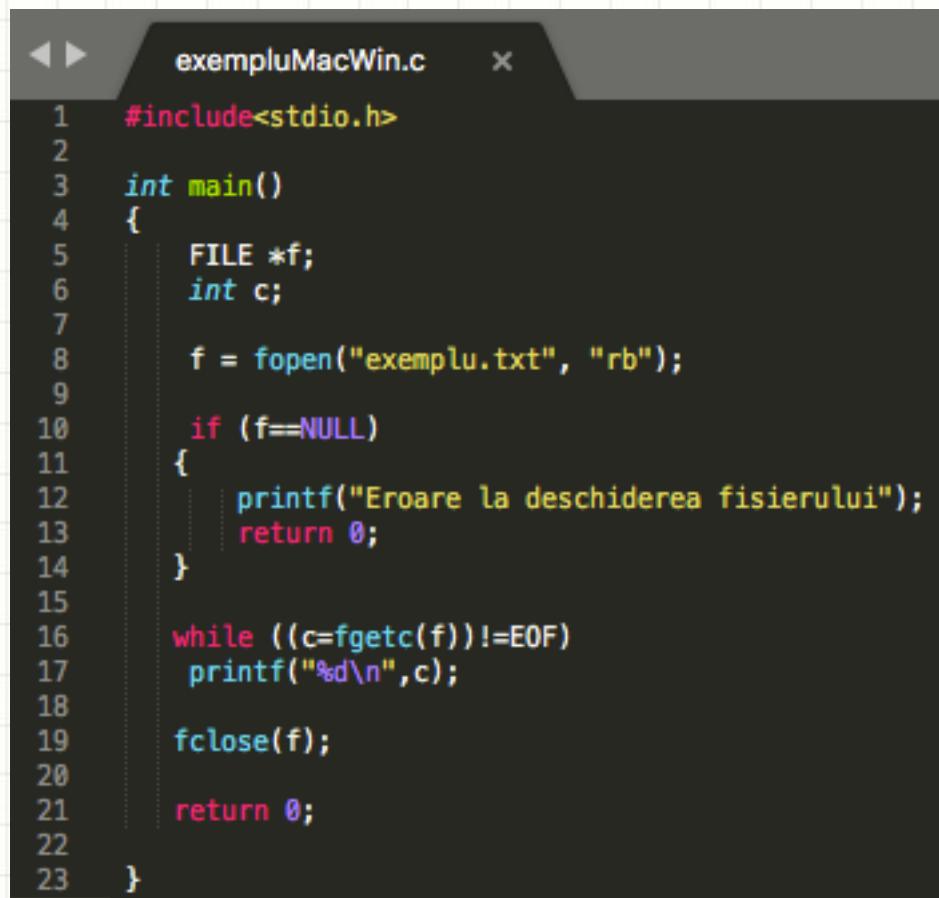
- **fișier binar = sir de octeți neformatat pe linii care este stocat pe suport magnetic/optic.**

- **FILE *fopen(char *nume_fisier, char *mod_deschidere)**
 - nume_fisier = numele fisierului
 - mod_deschidere = sir de caracter ce precizeaza tipul de acces la fisier:

Mod	Semnificație
r	Deschide un fisier tip text pentru a fi citit
w	Creează un fisier tip text pentru a fi scris
a	Adaugă într-un fisier tip text
rb	Deschide un fisier de tip binar pentru a fi citit
wb	Creează un fisier de tip binar pentru a fi scris
ab	Adaugă într-un fisier de tip binar
r+	Deschide un fisier tip text pentru a fi citit/scris
w+	Creează un fisier tip text pentru a fi citit/scris
a+	Adaugă în sau creează un fisier tip text pentru a fi citit/scris
r+b	Deschide un text în binar pentru a fi citit/scris
w+b	Creează un fisier de tip binar pentru a fi citit/scris
a+b	Adaugă sau creează un fisier de tip binar pentru a fi citit/scris

Fisiere binare

- fișier binar = sir de octeți neformatat pe linii care este stocat pe suport magnetic/optic.



The image shows a screenshot of a code editor window titled "exempluMacWin.c". The code is written in C and reads characters from a file named "exemplu.txt".

```
#include<stdio.h>
int main()
{
    FILE *f;
    int c;

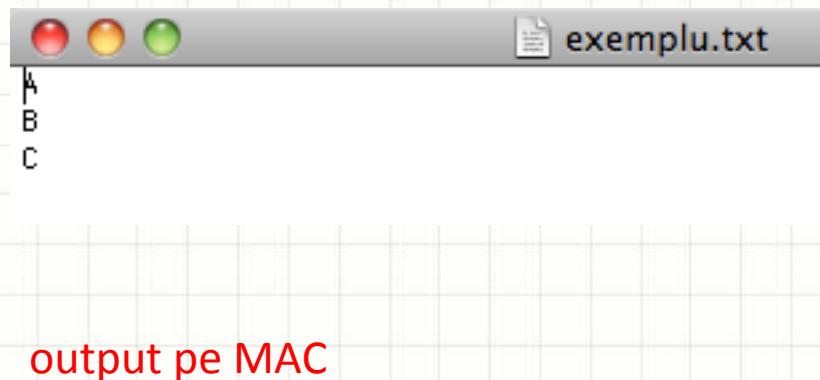
    f = fopen("exemplu.txt", "rb");
    if (f==NULL)
    {
        printf("Eroare la deschiderea fisierului");
        return 0;
    }

    while ((c=fgetc(f))!=EOF)
        printf("%d\n",c);

    fclose(f);
    return 0;
}
```

Fișiere binare

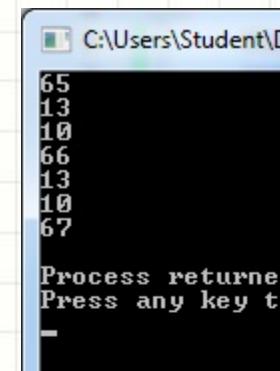
- fișier binar = sir de octeți neformatat pe linii care este stocat pe suport magnetic/optic. Octeții nu sunt considerați ca fiind coduri de caractere.



output pe MAC

```
Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ gcc exempluMacWin.c
Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ ./a.out
65
10
66
10
67
```

In Windows output-ul va arăta aşa încât CR+LF ('\r' + '\n') este terminator de linie (nu se translatează într-un LF ('\n'))



Functii de citire scriere

- `int fwrite(void *tablou, int dim_element, int nr_elem, FILE *f)`
 - scrie în fișierul referit de f cel mult nr_elem elemente de dimensiune dim_element de la adresa tablou;
- `int fread(void *tablou, int dim_element, int nr_elem, FILE *f)`
 - citește cel mult nr_elem elemente de dimensiune dim_element din fisierul referit de f la adresa tablou.

Functii de citire scriere

□ Exemplul 1



```
01 exempluFisierBinar1.c
02
03 #include<stdio.h>
04
05 int main()
06 {
07
08     FILE *f,*g;
09     int x = 1094861636;
10     f = fopen("numarBinar.out","wb");
11     g = fopen("numarText.out","w");
12     if ((f==NULL) || (g==NULL))
13     {
14         printf("Eroare la deschiderea fisierelor");
15         return 0;
16     }
17
18     fwrite(&x,sizeof(int),1,f);
19     fprintf(g,"%d",x);
20
21     fclose(f);
22     fclose(g);
23
24 }
```

Functii de citire scriere

Exemplul 1

```
exempluFisierBinari.c
00
01
02 #include<stdio.h>
03
04 int main()
05 {
06
07     FILE *f,*g;
08     int x = 1094861636;
09     f = fopen("numarBinar.out","wb");
10     g = fopen("numarText.out","w");
11     if ((f==NULL) || (g==NULL))
12     {
13         printf("Eroare la deschiderea fisierelor");
14         return 0;
15     }
16
17     fwrite(&x,sizeof(int),1,f);
18     fprintf(g,"%d",x);
19
20     fclose(f);
21     fclose(g);
22
23     return 0;
24 }
```



Octetii se reprezinta de la cel mai putin semnificativ la cel mai semnificativ pe masina mea (little endian)

Scrierea lui 1094861636 in baza 2:

0 | 1 | 0 | 0 | 0 | 0 | 0 | 1

0 | 1 | 0 | 0 | 0 | 0 | 1 | 0

0 | 1 | 0 | 0 | 0 | 0 | 1 | 1

0 | 1 | 0 | 0 | 0 | 1 | 0 | 0

65 = A

66 = B

67 = C

68 = D

Functii de citire scriere

Exemplul 2

```
exempluFisierBinar2.c  x

001 #include<stdio.h>
002
003 int main()
004 {
005
006     FILE *f;
007     int x;
008     f = fopen("numarBinar.out","rb");
009     if (f==NULL)
010     {
011         printf("Eroare la deschiderea fisierului");
012         return 0;
013     }
014     while(fread(&x,sizeof(int),1,f)==1)
015         printf("x=%d\n",x);
016     fclose(f);
017
018     char c;
019     f = fopen("numarBinar.out","rb");
020     while(fread(&c,sizeof(char),1,f)==1)
021         printf("c=%d\n",c);
022     fclose(f);
023
024     return 0;
025
026 }
```

```
Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ gcc exempluFisierBinar2.c
Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ ./a.out
x=1094861636
c=68
c=67
c=66
c=65
```

Functii de citire scriere

Exemplul 3

```
exempluFisierBinar3.c

1 #include<stdio.h>
2
3 int main()
4 {
5
6     FILE *f1,*f2,*f3;
7     f1 = fopen("numere1.txt","wb");
8     f2 = fopen("numere2.txt","wb");
9     f3 = fopen("numere3.txt","wb");
10    if ((f1==NULL) || (f2==NULL) || (f3==NULL))
11    {
12        printf("Eroare la deschiderea fisierelor");
13        return 0;
14    }
15    int v[5] = {33,40,50,10,80},i;
16    //varianta 1
17    for(i=0;i<5;i++)
18    {
19        fwrite(&v[i],sizeof(int),1,f1);
20    }
21    //varianta 2
22    fwrite(v,sizeof(int),5,f2);
23    //varianta 3
24    fwrite(v,5*sizeof(int),1,f3);
25
26    fclose(f1);
27    fclose(f2);
28    fclose(f3);
29    return 0;
30
31 }
```



Coduri ASCII:

33 = !, 40 = (, 50 = 2, 10 = \n, 80 = P

Functii de citire scriere

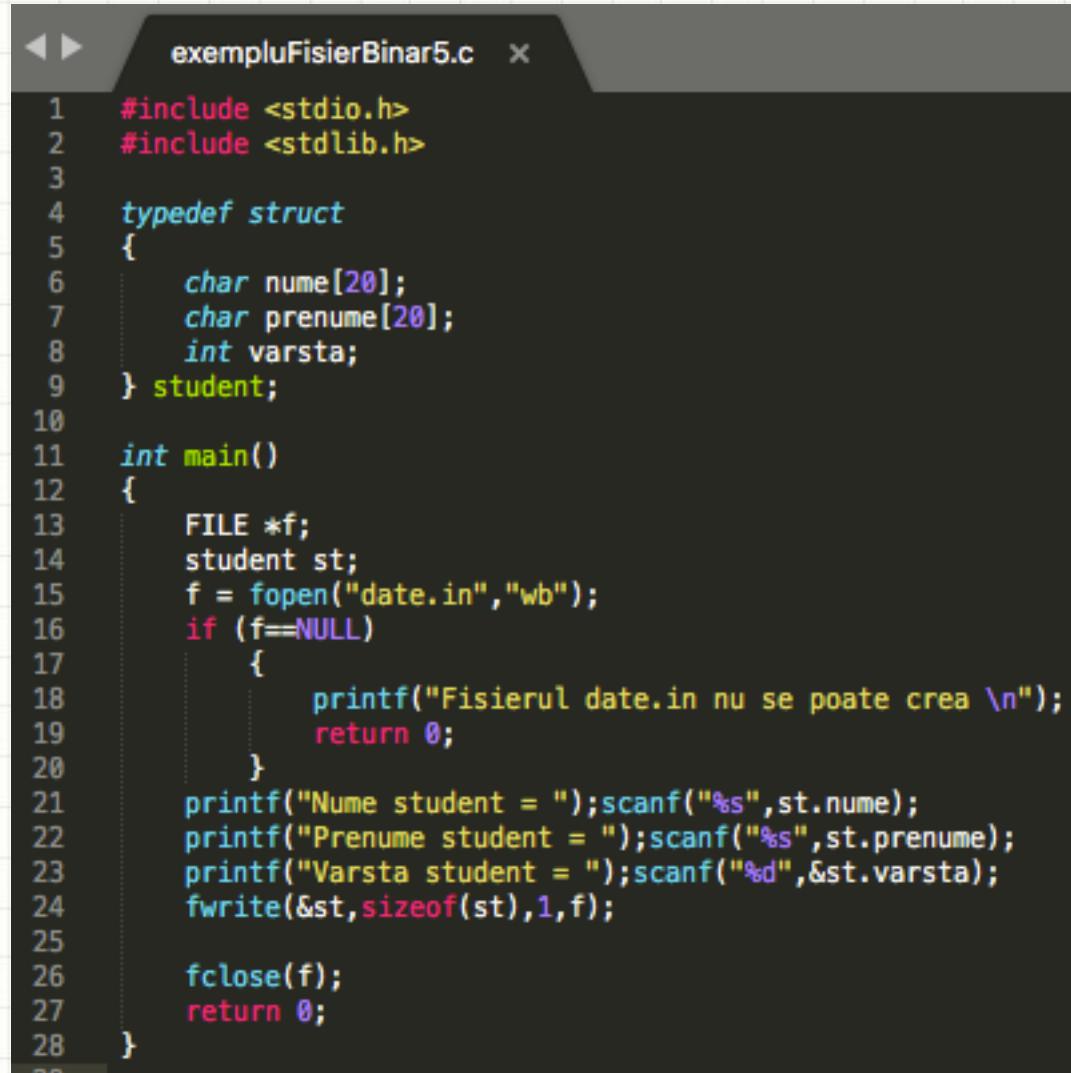
Exemplul 4

```
001 //include<stdio.h>
002
003 int main()
004 {
005
006     FILE *f1,*f2,*f3;
007     f1 = fopen("numere1.txt","rb");
008     f2 = fopen("numere2.txt","rb");
009     f3 = fopen("numere3.txt","rb");
010     if ((f1==NULL) || (f2==NULL) || (f3==NULL))
011     {
012         printf("Eroare la deschiderea fisierelor");
013         return 0;
014     }
015     int v[5],i;
016     for(i=0;i<5;i++)
017     {
018         fread(&v[i],sizeof(int),1,f2); printf("%d ",v[i]);
019     }
020     printf("\n");
021     //varianata 2
022     fread(v,sizeof(int),5,f3);
023     for(i=0;i<5;i++)
024         printf("%d ",v[i]);
025     printf("\n");
026     //varianata 3
027     fread(v,5*sizeof(int),1,f1);
028     for(i=0;i<5;i++)
029         printf("%d ",v[i]);
030     printf("\n");
031
032     fclose(f1);
033     fclose(f2);
034     fclose(f3);
035     return 0;
036 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ gcc exempluFisierBinar4.c
[Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ ./a.out
33 40 50 10 80
33 40 50 10 80
33 40 50 10 80
```

Functii de citire scriere

□ Exemplul 5: scrierea unei structuri



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

typedef struct
{
    char nume[20];
    char prenume[20];
    int varsta;
} student;

int main()
{
    FILE *f;
    student st;
    f = fopen("date.in","wb");
    if (f==NULL)
    {
        printf("Fisierul date.in nu se poate crea \n");
        return 0;
    }
    printf("Nume student = ");scanf("%s",st.nume);
    printf("Prenume student = ");scanf("%s",st.prenume);
    printf("Varsta student = ");scanf("%d",&st.varsta);
    fwrite(&st,sizeof(st),1,f);

    fclose(f);
    return 0;
}
```

Functii de citire scriere

Exemplul 5: scrierea unei structuri

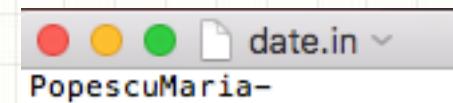
```
exempluFisierBinar5.c  x
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct
5 {
6     char nume[20];
7     char prenume[20];
8     int varsta;
9 } student;
10
11 int main()
12 {
13     FILE *f;
14     student st;
15     f = fopen("date.in","wb");
16     if (f==NULL)
17     {
18         printf("Fisierul date.in nu se poate
19             return 0;
20     }
21     printf("Nume student = ");scanf("%s",st.nume);
22     printf("Prenume student = ");scanf("%s",st.prenume);
23     printf("Varsta student = ");scanf("%d",&st.varsta);
24     fwrite(&st,sizeof(st),1,f);
25
26     fclose(f);
27     return 0;
28 }
```

```
Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ gcc exempluFisierBinar5.c
[Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ ./a.out
Nume student = Popescu
Prenume student = Maria
Varsta student = 20
```



Caracterul cu codul ASCII = 20 e
neprintabil

```
Bogdan-Alexes-MacBook-Pro:curs6 bogdan$ ./a.out
Nume student = Popescu
Prenume student = Maria
Varsta student = 45
```



Aplicație: procesarea de imagini

- ❑ o imagine este un fișier binar



Aplicație: procesarea de imagini

- ❑ o imagine este un fișier binar
- ❑ imagine color RGB
- ❑ fiecare pixel are o culoare dată de un triplet (r,g,b) cu valori pe fiecare canal intre 0 si 255 = 1 octet/canal
- ❑ albastru = (0,0 ,255), negru = (0,0,0),
rosu = (255, 0, 0), verde = (0,255,0)
- ❑ 1 pixel color = 3 canale = 3 octeți
- ❑ dimensiuni 600 x 800 pixeli
- ❑ $600 \times 800 \times 3$ octeți = 1440000 octeți



Aplicatie: procesarea de imagini

image.bmp Info

image.bmp 1.4 MB

Modified: Today, 00:14

Add Tags...

▼ General:

Kind: Windows bitmap image
Size: 1'440'054 bytes (1.4 MB on disk)
Where: Macintosh HD ▶ Users ▶ bogdan ▶ FMI ▶ PP ▶ Bogdan ▶ 2017_0212

Created: Today, 00:14
Modified: Today, 00:14

Stationery pad
 Locked

▼ More Info:

Where from: <http://cdn.instructables.com/ORIG/FWO/36LM/FLQAM1CS/FWO36LMFLQAM1CS.bmp>, <https://www.google.com/>

Dimensions: 800 × 600
Color space: RGB
Alpha channel: No

▼ Name & Extension:

image.bmp

Hide extension

Search



image.bmp

1.4 MB

Created Today, 00:14
Modified Today, 00:14
Last opened Today, 00:14
Dimensions 800 × 600

Add Tags...

Aplicație: procesarea de imagini

- ❑ o imagine este un fișier binar
- ❑ imagine color RGB
- ❑ fiecare pixel are o culoare dată de un triplet (r,g,b) cu valori pe fiecare canal între 0 și 255 = 1 octet/canal
- ❑ 1 pixel color = 3 canale = 3 octeți
- ❑ dimensiuni 600×800 pixeli
- ❑ $600 \times 800 \times 3$ octeți = 1440000 octeți
- ❑ 54 de octeți headerul + 144000 octeți pentru pixeli



modificaImagine.c x

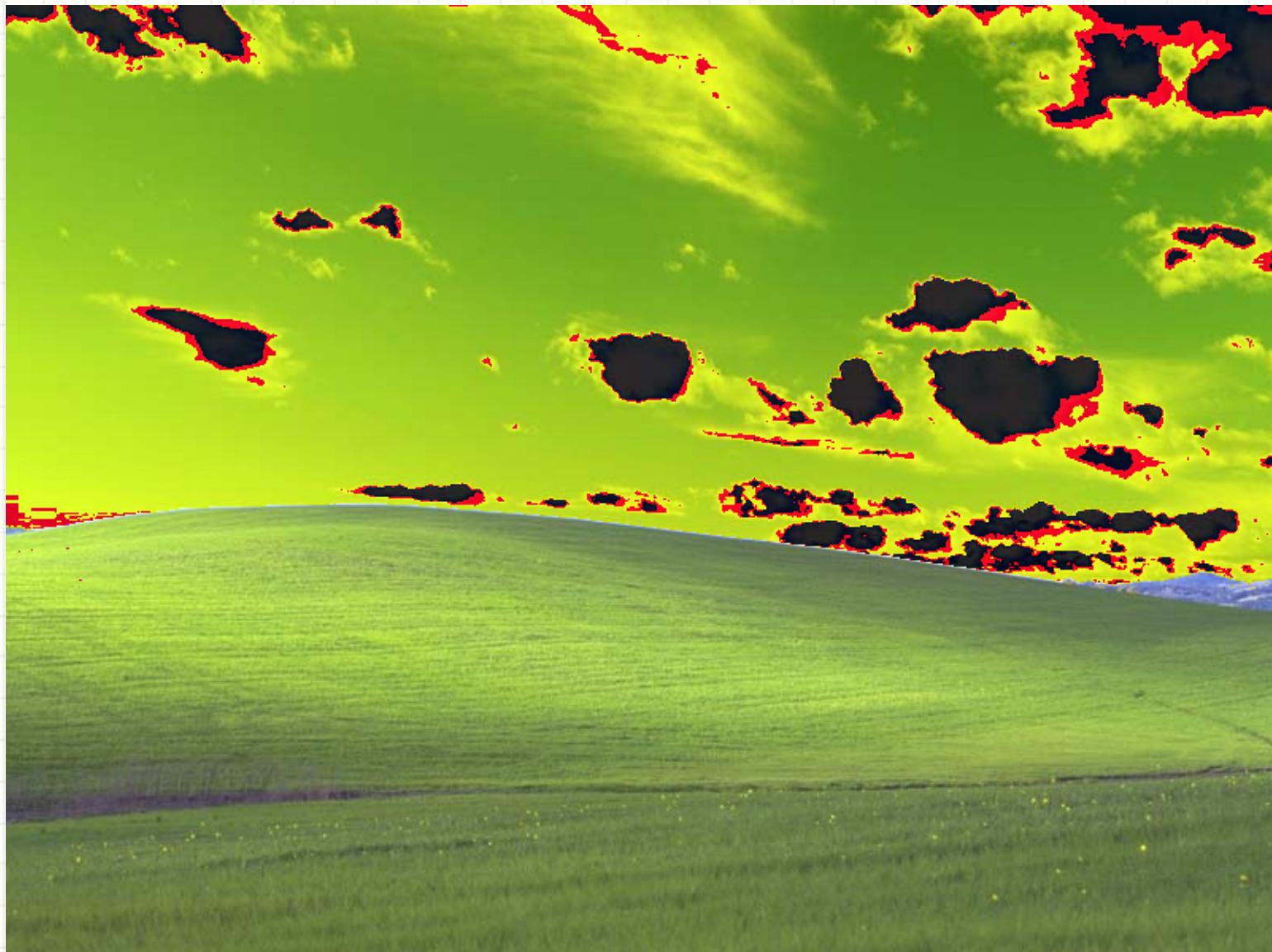
```
1 #include <stdio.h>
2
3 int main()
4 {
5     //pointeri de fisier
6     FILE *fin , *fout;
7
8     //deschid fisierul de intrare in mod binar (pentru citire)
9     fin = fopen("image.bmp" , "rb");
10
11    //deschid fisierul de iesire in mod binar (pentru scriere)
12    fout = fopen("image_modified.bmp" , "wb");
13
14    //copiez headerul
15    int i, x, y;
16    for(i = 0; i < 54; i++)
17    {
18        fread(&x , 1 , 1 , fin);
19        fwrite(&x , 1 , 1 , fout);
20    }
21
22    //citesc octet cu octet din fisierul de intrare
23    while(fread(&x , 1 , 1 , fin) == 1)
24    {
25        y = x + 50;
26        fwrite(&y , 1 , 1 , fout);
27    }
28
29    fclose(fin);
30    fclose(fout);
31
32    return 0;
33 }
```

Adaug 50 de unități la fiecare canal,

Schimb culorile în imagini

Cresc luminozitatea (brightness)

Ce obțin?



- ❑ pixeli negri = (0,0,0)
- ❑ de ce apar pixeli negri?
- ❑ $y = x + 50;$
- ❑ daca $x > 205$ atunci octetul scris din y va retine o valoare mică (apropiată de 0)



```
modificalmagine.c  x
1 #include <stdio.h>
2
3 int main()
4 {
5     //pointeri de fisier
6     FILE *fin , *fout;
7
8     //deschid fisierul de intrare in mod binar (pentru citire)
9     fin = fopen("image.bmp" , "rb");
10
11    //deschid fisierul de iesire in mod binar (pentru scriere)
12    fout = fopen("image_modified.bmp" , "wb");
13
14    //copiez headerul
15    int i, x, y;
16    for(i = 0; i < 54; i++)
17    {
18        fread(&x , 1 , 1 , fin);
19        fwrite(&x , 1 , 1 , fout);
20    }
21
22    //citesc octet cu octet din fisierul de intrare
23    while(fread(&x , 1 , 1 , fin) == 1)
24    {
25        y = x + 50;
26        if (y > 255)
27            y = 255;
28        fwrite(&y , 1 , 1 , fout);
29    }
30
31    fclose(fin);
32    fclose(fout);
```

Adaug 50 de unități la fiecare canal,

Schimb culorile în imagini

Cresc luminozitatea (brightness)

Ce obțin?





Imagine inițială



Imagine modificată

Cuprinsul cursului de azi

1. Fișiere: noțiuni generale.
2. Fișiere text: funcții specifice de manipulare.
3. Fișiere binare: funcții specifice de manipulare.
4. Funcții

Functii

- permit modularizarea programelor
 - variabilele declarate în interiorul funcțiilor – variabile locale (vizibile doar în interior)
- parametri funcțiilor
 - permit comunicarea informației între funcții
 - sunt variabile locale funcțiilor
- avantajele utilizării funcțiilor
 - divizarea problemei în subprobleme
 - managementul dezvoltării programelor
 - utilizarea/reutilizarea funcțiilor scrise în alte programe
 - elimină duplicarea codului scris

Functii

- o funcție = bloc de instrucțiuni care nu se poate executa de sine stătător ci trebuie apelat.

- sintaxa:

tip_returnat nume_functie (lista parametrilor formali)

```
{     variabile locale  
       instructiuni;  
       return expresie;  
}
```

antetul funcției
(declarare)

corpus funcției
(definire)

- lista de parametri formalii poate fi reprezentata de:
 - nici un parametru:
 - **tip_returnat nume_functie ()**
 - **tip_returnat nume_functie (void)**
 - unul sau mai mulți parametri separați prin virgulă.

Valoarea returnată de o funcție

- două categorii de funcții:
 - care returnează o valoare: prin utilizarea instrucțiunii **return expresie**;
 - care nu returnează o valoare: prin instrucțiunea **return**; (tipul returnat este void)
- returnarea valorii
 - poate returna orice tip standard (**void**, **char**, **int**, **float**, **double**) sau definit de utilizator (structuri, uniuni, enumerari)
 - declarațiile și instrucțiunile din funcții sunt executate până se întâlnește
 - instrucțiunea **return**
 - acolada închisă **}** - execuția atinge finalul funcției

Valoarea returnată de o funcție

```
double f(double t)
{
    return t-1.5;
}
```

← definire de funcție

```
float g(int);
```

← declarație de funcție

```
int main()
{
    float a=11.5;
    printf("%f\n", f(a));
    printf("%f\n", g(a));
}
```

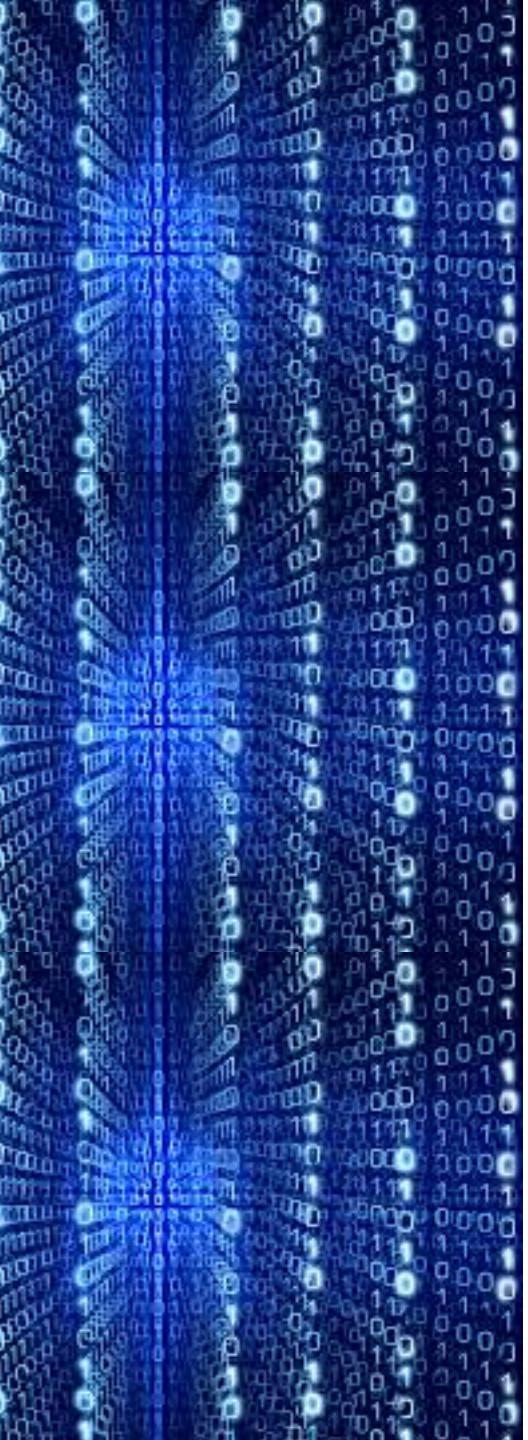
Rezultat afișat

10.000000

13.000000

```
float g(int z)
{
    return z+2.0;
}
```

← definire de funcție



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,

2018-2019

Cursul 7

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, siruri 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.
- Fișiere binare**
 - Funcții specifice de manipulare.
- Funcții (1)**
 - Declarație și definire. Apel. Metode de transmitere a parametrilor. Pointeri la funcții.
- Tablouri și pointeri**
 - Legătura dintre tablouri și pointeri
 - Aritmetică pointerilor
 - Alocarea dinamică a memoriei
 - Clase de memorare
- Siruri de caractere**
 - 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**

Cuprinsul cursului de azi

1. Funcții
2. Pointeri la funcții
3. Aritmetică pointerilor
4. Legătura dintre tablouri și pointeri

Functii

- o funcție = bloc de instrucțiuni care nu se poate executa de sine stătător ci trebuie apelat.

- sintaxa:

tip_returnat nume_functie (lista parametrilor formali)

```
{     variabile locale  
       instructiuni;  
       return expresie;  
}
```

antetul funcției
(declarare)

corpus funcției
(definire)

- lista de parametri formalii poate fi reprezentata de:
 - nici un parametru:
 - **tip_returnat nume_functie ()**
 - **tip_returnat nume_functie (void)**
 - unul sau mai mulți parametri separați prin virgulă.

Valoarea returnată de o funcție

- două categorii de funcții:
 - care returnează o valoare: prin utilizarea instrucțiunii **return expresie**;
 - care nu returnează o valoare: prin instrucțiunea **return**; (tipul returnat este void)
- returnarea valorii
 - poate returna orice tip standard (**void**, **char**, **int**, **float**, **double**) sau definit de utilizator (structuri, uniuni, enumerari)
 - declarațiile și instrucțiunile din funcții sunt executate până se întâlnește
 - instrucțiunea **return**
 - acolada închisă **}** - execuția atinge finalul funcției

Valoarea returnată de o funcție

```
double f(double t)
{
    return t-1.5;
}
```

← definire de funcție

```
float g(int);
```

← declarație de funcție

```
int main()
{
    float a=11.5;
    printf("%f\n", f(a));
    printf("%f\n", g(a));
}
```

Rezultat afișat

10.000000

13.000000

```
float g(int z)
{
    return z+2.0;
}
```

← definire de funcție

Prototipul și argumentele funcțiilor

- **prototipul** unei funcții (declararea ei) constă în specificarea antetului urmat de caracterul ;
 - nu este necesară specificarea numelor parametrilor formali
int adunare(int, int);
 - este necesară inserarea prototipului unei funcții înaintea altor funcții în care este invocată dacă definirea ei este localizată după definirea acestor funcții
- **parametri** apar în definiții
- **argumentele** apar în apelurile de funcții
 - corespondența între parametrii formali (definiția funcției) și actuali (apelul funcției) este **pozițională**
 - regula de **conversie a argumentelor**
 - în cazul în care diferă, tipul fiecărui argument este convertit automat la tipul parametrului formal corespunzător (ca și în cazul unei simple atribuirii)

Transmiterea parametrilor către funcții

- ❑ utilizată la apelul funcțiilor
- ❑ În limbajul C transmiterea parametrilor se poate face doar prin **valoare (pass-by-value)**
 - ❑ o copie a argumentelor este trimisă funcției
 - ❑ modificările în interiorul funcției nu afectează argumentele originale
- ❑ În limbajul C++ transmiterea parametrilor apelul se poate face și prin **referință (pass-by-reference)**
 - ❑ argumentele originale sunt trimise funcției
 - ❑ modificările în interiorul funcției afectează argumentele trimise

Cod scris în C++ !!!

```
interschimbare.cpp  x

1 #include <stdio.h>
2
3 void interschimba1(int x, int y)
4 {
5     int aux = x; x = y; y = aux;
6 }
7
8 void interschimba2(int& x, int& y)
9 {
10    int aux = x; x = y; y = aux;
11 }
12
13 void interschimba3(int* x, int* y)
14 {
15     int aux = *x; *x = *y; *y = aux;
16 }
17
18 int main()
19 {
20     int x = 10, y = 15;
21     interschimba1(x,y);
22     printf("x = %d, y = %d \n",x,y); <-- apel prin valoare
23
24     x = 10, y = 15;
25     interschimba2(x,y);
26     printf("x = %d, y = %d \n",x,y); <-- apel prin referință
27     numai în C++
28
29     x = 10, y = 15;
30     interschimba3(&x,&y);
31     printf("x = %d, y = %d \n",x,y); <-- apel prin valoare (se
32     transmit adresele variabilelor = pointeri)
33 }
```

```
Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ g++ interschimbare.cpp
Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
x = 10, y = 15
x = 15, y = 10
x = 15, y = 10
```

apel prin valoare

apel prin referință
numai în C++

apel prin valoare (se
transmit adresele
variabilelor = pointeri)

Transmiterea parametrilor către funcții

- ❑ utilizat la apelul funcțiilor
- ❑ În limbajul C transmiterea parametrilor se poate face doar prin **valoare (pass-by-value)**
 - ❑ o copie a argumentelor este trimisă funcției
 - ❑ modificările în interiorul funcției nu afectează argumentele originale
- ❑ **nu există apel prin referință în limbajul C**
- ❑ pentru modificarea parametrilor actuali, funcției i se transmit nu valorile parametrilor actuali, ci **adresele lor (pass by pointer)**. Funcția face o copie a adresei dar prin intermediul ei lucrează cu variabila “reală” (zona de memorie “reală”). Astfel **putem simula în C transmiterea prin referință cu ajutorul pointerelor.**

Transmiterea parametrilor către funcții

```
exempluTransmitere1.c  x

1 #include <stdio.h>
2
3 int f1(int a, int b)
4 {
5     a++;
6     b++;
7     printf("In f1 avem a= %d \t b = %d \n",a,b);
8     return a + b;
9 }
10
11 int f2(int *a, int b)
12 {
13     *a = *a + 1;
14     b++;
15     return *a + b;
16 }
17
18 int main()
19 {
20     int x = 5, y = 8;
21     int z = f1(x, y);
22     printf("In main dupa f1 avem x = %d, y = %d, z = %d \n",x,y,z);
23     z = f2(&x, y);
24     printf("In main dupa f1 avem x = %d, y = %d, z = %d \n",x,y,z);
25     return 0;
26 }
```

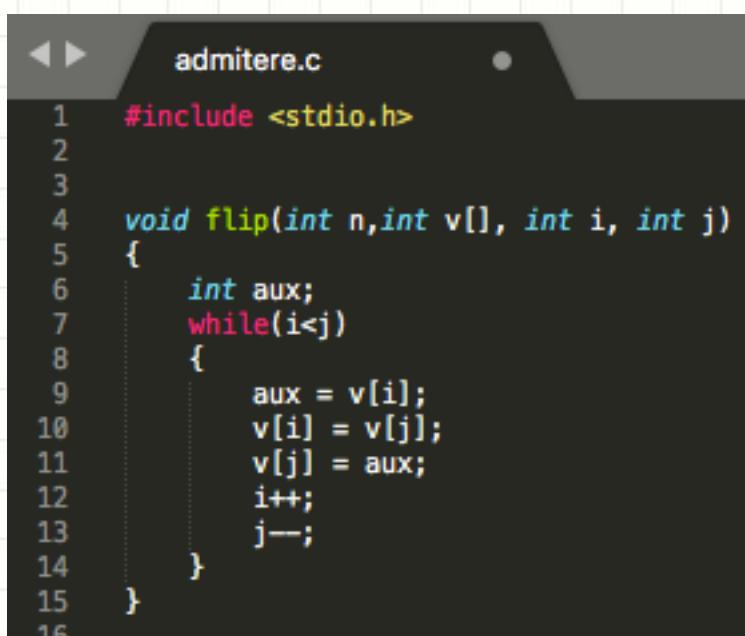
```
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc exempluTransmitere1.c
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
In f1 avem a= 6          b = 9
In main dupa f1 avem x = 5, y = 8, z = 15
In main dupa f1 avem x = 6, y = 8, z = 15
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ]
```

Problema de la admitere iunie 2017

IV. Informatică.

Fie n un număr natural nenul. Fie v un vector cu n poziții numerotate de la 1 la n și elemente numere naturale diferite, de la 1 la n , într-o ordine oarecare. Pentru i și j numere naturale între 1 și n , numim $\text{FLIP}(n, v, i, j)$ operația care inversează ordinea elementelor din v situate pe pozițiile de la i la j .

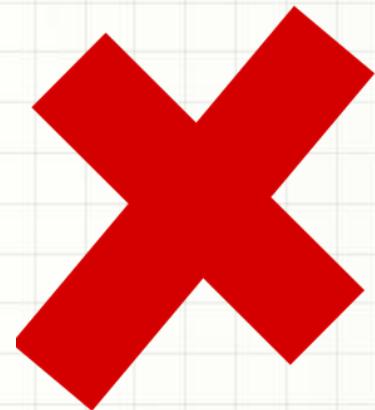
- a) Să se scrie în limbaj de programare o procedură (sau funcție) care implementează operația $\text{FLIP}(n, v, i, j)$.



```
admitere.c
001 #include <stdio.h>
002
003
004 void flip(int n,int v[], int i, int j)
005 {
006     int aux;
007     while(i<j)
008     {
009         aux = v[i];
010         v[i] = v[j];
011         v[j] = aux;
012         i++;
013         j--;
014     }
015 }
```

Rezolvare 1

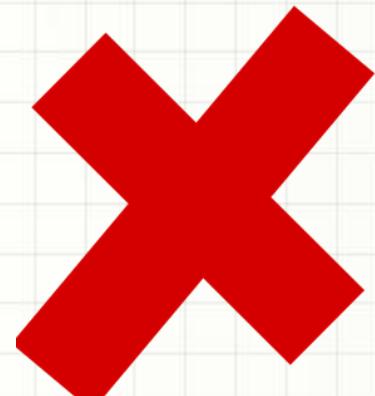
```
a) void FLIP (int n, int v[], int i, int j)
    {
        if (i < j)
            {
                int aux = v[i];
                v[i] = v[j];
                v[j] = aux;
            }
        FLIP (n, v, i+1, j-1);
    }
```



Dacă pun apelul recursiv în IF e soluție bună, altfel ieșe din vector la stânga și la dreapta

Rezolvare 2

```
a. void FLIP(int n, int V[n], int c, j
void FLIP(int n, int V[n], int c, int j)
{
    int aux;
    for(j; j >= c; j--)
        for(c; c <= j; c++)
    {
        V[j] = V
        aux = V[j];
        V[j] = V[c];
        V[c] = aux;
    }
}
```



La prima iteratie îl ajunge pe j, apoi algoritmul se opreste

Rezolvare 3

Subiectul IV

a) void FLIP(unsigned n, unsigned v[101], unsigned i,
unsigned j)

```
{ unsigned aux=0, nr=0;  
for( int l=i; l<=j; l++)  
{ aux=v[l];  
v[l]=v[j-nr];  
v[j-nr]=aux;  
nr++;  
if (nr===(j-i+1)/2)  
{ l=j; } }
```

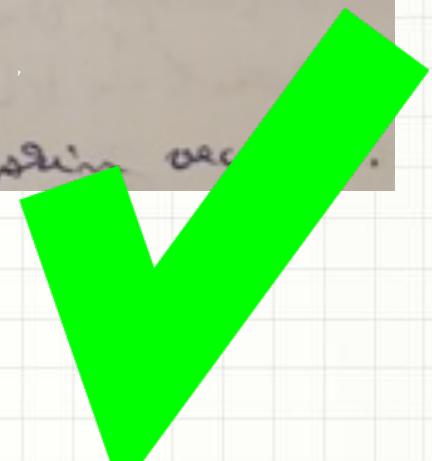


Rezolvare 4

IV Informatică

a) void FLIP(int n, int v[1000], int i, int j)
{ int k; aux;
for(k = i; k <= i + (j - i - 1)/2; k++)
{ aux = v[k];
v[k] = v[j + i - k];
v[j + i - k] = aux; } }

g. să se scrie o funcție care să înlocuiască elementele de la index i la j.



Returnarea de valori multiple

- ❑ o funcție returnază/întoarce maxim o singură valoare;
- ❑ putem întoarce (modifica) mai multe valori fie:
 - ❑ transmițând multiple variabile prin pointeri;
 - ❑ folosind tipuri de date derivate (structuri);
 - ❑ combinații (pointeri + valori întoarse);
- ❑ exemplu: funcție care calculează maximul și minimul valorilor unui tablou v unidimensional cu n numere întregi

Returnarea de valori multiple

- exemplu: funcție care calculează maximul și minimul valorilor unui tablou v unidimensional cu n numere întregi

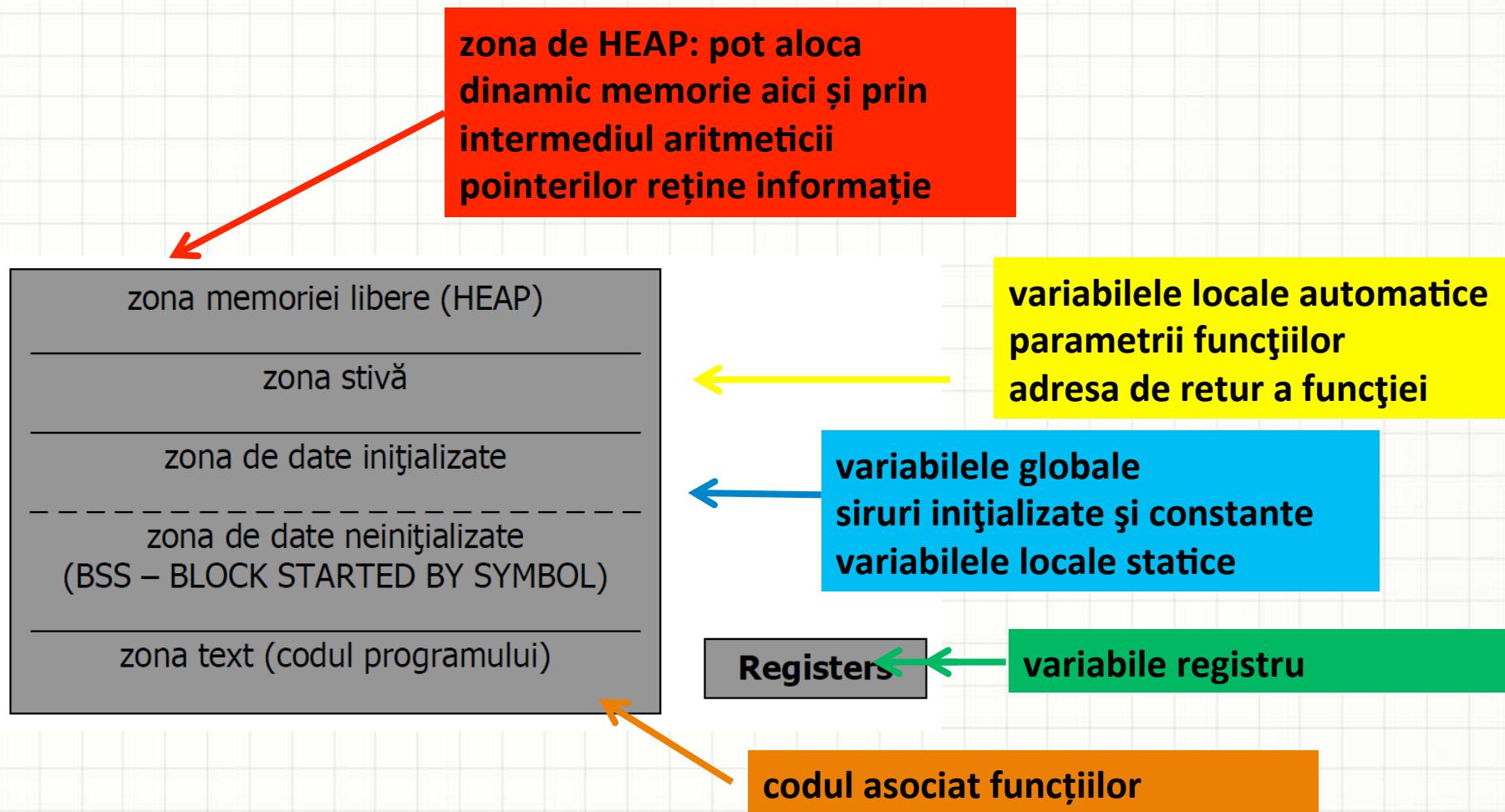
```
#include <stdio.h>
typedef struct
{
    int min;
    int max;
} minmax;
minmax calculeazaMinimMaximTablou1(int v[], int n)
{
    int minim = v[0];
    int maxim = v[0];
    for(int i = 1;i < n;i++)
    {
        if (minim > v[i])
            minim = v[i];
        if (maxim < v[i])
            maxim = v[i];
    }
    minmax x;
    x.min = minim;
    x.max = maxim;
    return x;
}
```

Returnarea de valori multiple

- exemplu: funcție care calculează maximul și minimul valorilor unui tablou v unidimensional cu n numere întregi

```
27 void calculeazaMinimMaximTablou2(int v[],int n, int* min,int *max)
28 {
29     *min = v[0];
30     *max = v[0];
31     for(int i = 1;i < n;i++)
32     {
33         if (*min > v[i])
34             *min = v[i];
35         if (*max < v[i])
36             *max = v[i];
37     }
38 }
39
40 int calculeazaMinimMaximTablou3(int v[],int n, int* min)
41 {
42     *min = v[0];
43     int max = v[0];
44     for(int i = 1;i < n;i++)
45     {
46         if (*min > v[i])
47             *min = v[i];
48         if (max < v[i])
49             max = v[i];
50     }
51     return max;
52 }
```

Harta simplificată a memoriei la rularea unui program



Harta simplificată a memoriei la rularea unui program

```
hartaMemorie.c
```

```
1 #include <stdio.h>
2
3 // variabile globale neinitialize
4 int g1,g2;
5
6 // variabile globale initialize
7 int g3=5, g4 = 7;
8
9 // variabile globale neinitialize
10 int g5, g6;
11
12 void f1() {
13     int var1, var2;
14     printf("In Stiva prin f1:\t\t %p %p\n",&var1,&var2);
15 }
16
17 void f2() {
18     int var1, var2;
19     printf("In Stiva prin f2:\t\t %p %p\n",&var1,&var2);
20     f1();
21 }
22
23 int main() {
24     //variabile locale
25     int var1, var2;
26     printf("In Stiva prin main:\t\t %p %p\n",&var1,&var2);
27     f2();
28     // variabile globale initialize + neinitialize
29     printf("Variabile globale neinitialize: %p %p\n",&g1,&g2);
30     printf("Variabile globale initialize:    %p %p\n",&g3,&g4);
31     printf("Variabile globale neinitialize: %p %p\n",&g5,&g6);
32     //cod
33     printf("Text Data:\t\t %p %p \n\n",main,f1);
34     return 0;
35 }
```

Harta simplificată a memoriei la rularea unui program

```
hartaMemorie.c
```

```
1 #include <stdio.h>
2
3 // variabile globale neinitialize
4 int g1,g2;
5
6 // variabile globale initialize
7 int g3=5, g4 = 7;
8
9 // variabile globale neinitialize
10 int g5, g6;
11
12 void f1() {
13     int var1, var2;
14     printf("In Stiva prin f1:\t\t %p %p\n",&var1,&var2);
15 }
16
17 void f2() {
18     int var1, var2;
19     printf("In Stiva prin f2:\t\t %p %p\n",&var1,&var2);
20     f1();
21 }
22
23 int main() {
24     //variabile locale
25     int var1, var2;
26     printf("In Stiva prin main:\t\t %p %p\n",&var1,&var2);
27     f2();
28     // variabile globale initialize
29     printf("Variabile globale initialize: %p %p\n",&g1,&g2);
30     printf("Variabile globale neinitialize: %p %p\n",&g3,&g4);
31     printf("Variabile globale neinitialize: %p %p\n",&g5,&g6);
32     //cod
33     printf("Text Data:\t\t\t %p %p\n",&g1,&g2);
34     return 0;
35 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc hartaMemorie.c
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
In Stiva prin main:          0x7fff532fab58 0x7fff532fab54
In Stiva prin f2:            0x7fff532fab1c 0x7fff532fab18
In Stiva prin f1:            0x7fff532faafc 0x7fff532faaf8
Variabile globale initialize: 0x10c906020 0x10c906024
Variabile globale neinitialize: 0x10c906018 0x10c90601c
Variabile globale neinitialize: 0x10c906028 0x10c90602c
Text Data:                  0x10c905e10 0x10c905db0
```

Stiva în C

- ❑ la execuția programelor C se utilizează o structură internă numită **stivă** și care este utilizată pentru alocarea memoriei și manipularea variabilelor temporare
- ❑ pe stivă sunt alocate și memorate:
 - ❑ variabilele locale din cadrul funcțiilor
 - ❑ parametrii funcțiilor
 - ❑ adresele de return ale funcțiilor
- ❑ dimensiunea implicită a stivei este redusă
 - ❑ în timpul execuției programele trebuie să nu depășească dimensiunea stivei
 - ❑ dimensiunea stivei poate fi modificată în prealabil din setările editorului de legături (*linker*)

Apelul funcției și revenirea din apel

- etapele principale ale apelului unei funcție și a revenirii din acesta în funcția de unde a fost apelată:
 - argumentele apelului sunt evaluate și trimise funcției
 - adresa de revenire este salvată pe stivă
 - controlul trece la funcția care este apelată
 - funcția apelată alocă pe **stivă** spațiu pentru variabilele locale
 - se execută instrucțiunile din corpul funcției
 - dacă există valoare returnată, aceasta este pusă într-un loc sigur
 - spațiul alocat pe stivă este eliberat
 - utilizând adresa de revenire controlul este transferat în funcția care a inițiat apelul, după acesta

Stiva în C – depășirea dimensiunii

- ambele programe eşuează în timpul execuției din cauza depășirii dimensiunii stivei

```
exempluStiva1.c
```

```
1 #include <stdio.h>
2
3 int f()
4 {
5     int a[10000000] = {0};
6     return 1;
7 }
8
9 int main()
10 {
11     f();
12     return 0;
13 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc exempluStiva1.c
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
Segmentation fault: 11
```

```
exempluStiva2.c
```

```
1 #include <stdio.h>
2
3
4 int f(int a,int b)
5 {
6     printf("a = %d, b = %d \n", a, b);
7     if (a<b)
8         return 1+f(a+1,b-1);
9     return 0;
10 }
11
12 int main()
13 {
14     printf("%d",f(0,1000000));
15     return 0;
16 }
```

```
a = 262048, b = 737952
a = 262049, b = 737951
a = 262050, b = 737950
a = 262051, b = 737949
Segmentation fault: 11
```

Cuprinsul cursului de azi

1. Funcții
2. Pointeri la funcții
3. Aritmetică pointerilor
4. Legătura dintre tablouri și pointeri

Harta simplificată a memoriei la rularea unui program

```
hartaMemorie.c x

1 #include <stdio.h>
2
3 // variabile globale neinitializate
4 int g1,g2;
5
6 // variabile globale initialize
7 int g3=5, g4 = 7;
8
9 // variabile globale neinitializate
10 int g5, g6;
11
12 void f1() {
13     int var1, var2;
14     printf("In Stiva prin f1:\t\t%p %p\n",&var1,&var2);
15 }
16
17 void f2() {
18     int var1, var2;
19     printf("In Stiva prin f2:\t\t%p %p\n",&var1,&var2);
20     f1();
21 }
22
23 int main() {
24     //variabile locale
25     int var1, var2;
26     printf("In Stiva prin main:\t\t%p %p\n",&var1,&var2);
27     f2();
28     // variabile globale initialize + neinitializate
29     printf("Variabile globale neinitializate: %p %p\n",&g1,&g2);
30     printf("Variabile globale initialize:    %p %p\n",&g3,&g4);
31     printf("Variabile globale neinitializate: %p %p\n",&g5,&g6);
32     //cod
33     printf("Text Data:\t\t\t%p %p \n\n",main,f1); ←
34     return 0;
35 }
```

Numele unei funcții neînsoțit de o listă de argumente este adresa de început a codului funcției și este interpretat ca un pointer la funcția respectivă

Pointeri la funcții

- pointer la o funcție = variabilă ce stochează adresa de început a codului asociat funcției
- sintaxa: **tip (*nume_pointer_functie) (tipuri argumente)**
 - **tip** = tipul de bază returnat de funcția spre care pointeaza nume_pointer_functie
 - **nume_pointer_functie** = variabila de tip pointer la o functie care poate lua ca valori adrese de memorie unde începe codul unei funcții
 - **observație:** trebuie să pun paranteză în definiție altfel definesc o funcție care întoarce un pointer de un anumit tip de date
- exemplu:
 - void (*pf)(int)
 - int (*pf)(int, int)
 - double (*pf)(int, double*)

Pointeri la funcții

```
exempluPointeriFunctii.c  x

1 #include <stdio.h>
2
3 int suma(int a, int b)
4 {
5     return a + b;
6 }
7
8 int diferență(int a, int b)
9 {
10    return a - b;
11 }
12
13 int main()
14 {
15     int (*pf)(int,int);
16     pf = &suma;
17     int s = (*pf)(2,5);
18     printf("s = %d \n",s);
19     pf = diferență;
20     int d = pf(2,5);
21     printf("d = %d \n",d);
22     return 0;
23 }
```

1. pentru a asigura unui pointer adresa unei functii, trebuie folosit numele functiei fara paranteze.
2. numele unei functii este un pointer spre adresa sa de inceput din segmentul de cod: f==&f

```
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc exempluPointeriFunctii.c
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
s = 7
d = -3
```

Utilitatea pointerilor la funcții

- se folosesc în **programarea generică**, realizăm apeluri de tip **callback**;
- o funcție C transmisă, printr-un pointer, ca argument unei alte funcții F se numește și funcție **“callback”**, pentru că ea va fi apelată “înapoi” de funcția F
- **exemple:**
 1. int suma(int n, int (*expresie)(int)); **(sumă generică de n numere)**
 2. void qsort(void *adresa,int nr_elemente, int dimensiune_element, int (*cmp)(const void *, const void *)); **(funcția qsort din stdlib.h)**

Utilitatea pointerilor la funcții

- exemplul 1: vreau să calculez suma

$$S_k(n) = \sum_{i=1}^n i^k$$

$$S_1(n) = 1 + 2 + \dots + n$$

$$S_2(n) = 1^2 + 2^2 + \dots + n^2$$

$$S_k(n) = \sum_{i=1}^n expresie(i)$$

Folosind pointeri la funcții pot să văd funcția ca o variabilă

Utilitatea pointerilor la funcții

- exemplul 1: vreau să calculez suma
- implementare elegantă:

$$S_k(n) = \sum_{i=1}^n i^k$$

```
int suma(int n, int (*expresie)(int))
{
    int i, s = 0;
    for (i = 1; i <= n; i++)
        s = s + expresie(i);
    return s;
}
```

```
int expresie1(int x)
{
    return x;
}
```

```
int expresie2(int x)
{
    return x*x;
}
```

Utilitatea pointerilor la funcții

□ exemplul 1: vreau să calculez suma

```
exempluPointeriFunctii2.c  x

1 #include <stdio.h>
2
3 int suma(int n, int (*expresie)(int))
4 {
5     int i, s = 0;
6     for (i = 1; i <= n; i++)
7         s = s + expresie(i);
8     return s;
9 }
10
11 int expresie1(int x)
12 {
13     return x;
14 }
15
16 int expresie2(int x)
17 {
18     return x*x;
19 }
20
21 int main()
22 {
23     int S1 = suma(5,expresie1);
24     printf("S1 = %d\n",S1);
25     int S2 = suma(5,expresie2);
26     printf("S2 = %d\n",S2);
27     return 0;
28 }
```

$$S_k(n) = \sum_{i=1}^n i^k$$

```
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc exempluPointeriFunctii2.c
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
```

S1 = 15

S2 = 55

Utilitatea pointerilor la funcții

- exemplul 2: funcția qsort din stdlib.h folosită pentru sortarea unui vector/tablou. Antetul lui qsort este:

```
void qsort (void *adresa, int nr_elemente, int dimensiune_element,  
int (*cmp) (const void *, const void *))
```

- adresa = pointer la adresa primului element al tabloului ce urmeaza a fi sortat
(pointer generic – nu are o aritmetică inclusă)
- nr_elemente = numarul de elemente al vectorului
- dimensiune_element = dimensiunea in octeți a fiecărui element al tabloului
(char = 1 octet, int = 4 octeți, etc)
- cmp – funcția de comparare a două elemente

Functia qsort (și în cursul 4)

```
void qsort (void *adresa, int nr_elemente, int dimensiune_element,  
int (*cmp) (const void *, const void *))  
  
int cmp(const void *a, const void *b)
```

adresele a două elemente din tablou

Cmp este o funcție generică comparator, compară 2 elemente de orice tip din tablou. Întoarce:

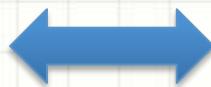
- un număr < 0 dacă vrem elementul de la adresa **a** la stânga (înaintea) elementului de la adresa **b**
- un număr >0 dacă vrem elementul de la adresa **a** la dreapta (după) elementului de la adresa **b**
- 0, dacă nu contează

Funcția qsort pentru întregi

```
void qsort (void *adresa, int nr_elemente, int dimensiune_element,  
int (*cmp) (const void *, const void *))
```

Exemplu de funcție cmp pentru sortarea unui vector de numere
întregi:

```
int cmp(const void* a, const void *b)  
{  
    int va, vb;  
    va = *(int*)a;  
    vb = *(int*)b;  
    if(va < vb) return -1;  
    if(va > vb) return 1;  
    return 0;  
}
```



```
int cmp(const void* a, const void *b)  
{  
    return *(int*)a - *(int*)b;  
}
```

Funcția qsort pentru întregi

```
exempluqsort.c
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int cmp(const void* a, const void *b)
5 {
6     return *(int*)a - *(int*)b;
7 }
8
9 int main()
10 {
11     int v[] = {0, 5, -6, 9, 7, 12, 8, 7, 4};
12     qsort(v, 9, sizeof(int), cmp);
13     for( int i = 0; i < sizeof(v)/sizeof(int); i++)
14         printf("%d \t", v[i]);
15     printf("\n");
16     return 0;
17 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs5 bogdan$ ./a.out
```

```
-6      0      4      5      7      7      8      9      12
```

Funcția qsort pentru structuri

Exercițiu seminar: avem o structură care reține numele, prețul, cantitatea pentru fiecare produs dintr-un magazin. Se dă un vector de asemenea produse pe care vreau să îl sorteze descrescător după preț și în caz de prețuri egale în ordinea alfabetică a numelor produselor.

```
produs.c

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 struct Produs
6 {
7     char nume[25];
8     double pret;
9     double cantitate;
10 };
11
12
13 int cmpProduse(const void *a, const void* b)
14 {
15     if (((struct Produs *)a)->pret == ((struct Produs *)b)->pret)
16         return strcmp(((struct Produs *) a)->nume,((struct Produs *)b)->nume);
17
18     if (((struct Produs *)a)->pret > ((struct Produs *)b)->pret)
19         return -1;
20     return +1;
21 }
22 }
```

Exercițiu seminar 5

Să se calculeze integrala definită (între a și b) a unei funcții f: R → R

$$\int_a^b f(x) dx$$

```
double calculeazaIntegralaDefinita(double a, double b, double precizie, double (*pf)(double))
```

Soluție: calculez suma Riemann superioară și inferioară pe o diviziune cu un pas foarte mic până când diferența dintre cele două sume este mai mică decât precizia pe care o doresc.

Cuprinsul cursului de azi

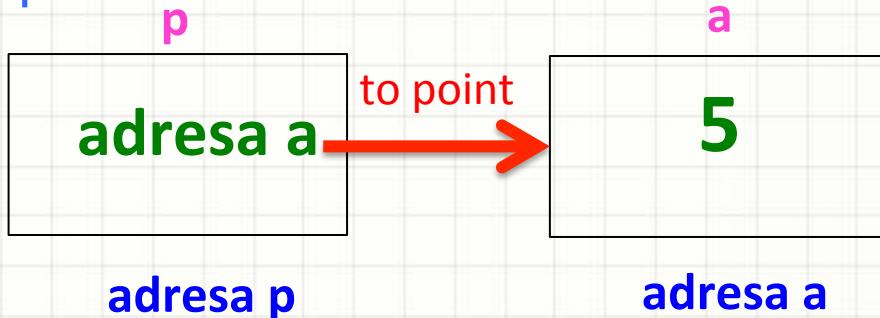
1. Funcții
2. Pointeri la funcții
3. Aritmetică pointerilor
4. Legătura dintre tablouri și pointeri

Aritmetica pointerilor

- un pointer: variabilă care poate stoca adrese de memorie

- exemple:

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



- asupra pointerilor pot fi realizate operații aritmetice:

- incrementare (++), decrementare (--);
 - adăugare (+ sau +=) sau scădere a unui intreg (- sau -=)
 - scădere a unui pointer din alt pointer;
 - asignări;
 - comparații.

Aritmetica pointerilor

- inițializarea unui pointer cu adresa primul element al unui tablou

```
aritmeticaPointeri1.c
#include <stdio.h>
int main()
{
    int v[5];
    int *p;

    p = &v[0];
    printf("Adresa lui v[0] este %p \n", p);
    printf("Valoarea lui v este %p \n", v);
    printf("Adresa lui v este %p \n", &v);

    if (p==v)
        printf("Acleeasi adresa \n");

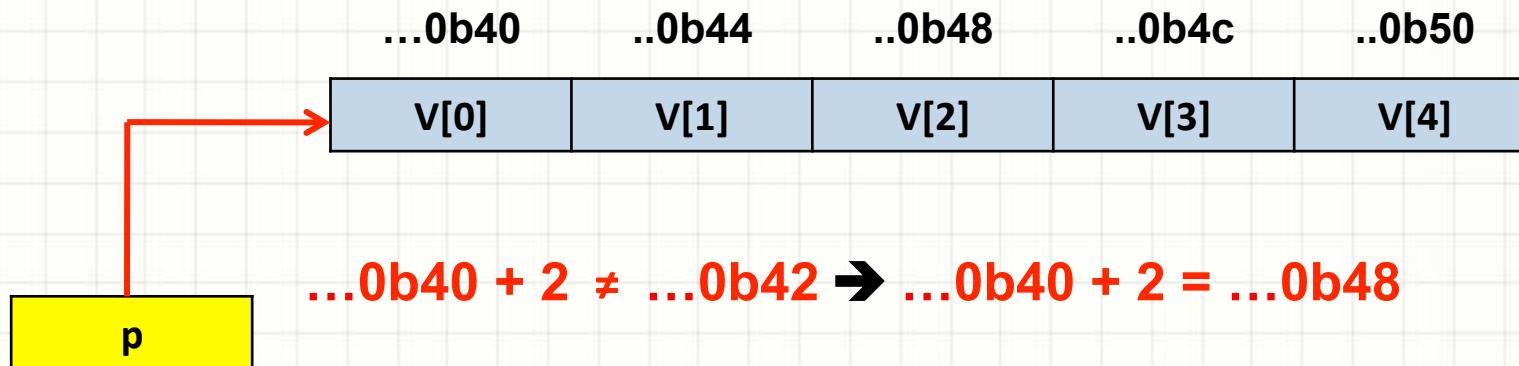
    return 0;
}
```

v este un pointer care
pointeaza către v[0]

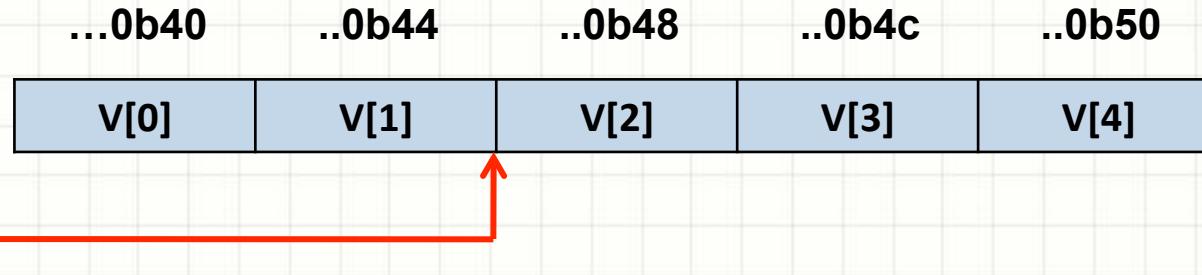
```
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc aritmeticaPointeri1.c
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
Adresa lui v[0] este 0x7fff5e600b40
Valoarea lui v este 0x7fff5e600b40
Adresa lui v este 0x7fff5e600b40
Acleeasi adresa
```

Aritmetică pointerilor

- adunarea/scăderea unui număr natural dintr-un pointer



- în aritmetică pointerilor adăugarea unui întreg la o adresă de memorie are ca rezultat o nouă adresă de memorie!



Aritmetica pointerilor

- adunarea/scăderea unui număr natural dintr-un pointer

```
aritmeticaPointeri2.c  x

1 #include <stdio.h>
2
3 int main()
4 {
5     int v[5];
6     int *p;
7
8     p = &v[0];
9     printf("Adresa lui v[0] este %p \n", p);
10    printf("Valoarea lui v este %p \n",v);
11    printf("Adresa lui v este %p \n",&v);
12
13    if (p==v)
14        printf("Aceeași adresa \n");
15
16    printf("%p \n %p \n %p \n %p \n %p \n", v,v+1,v+2,v+3,v+4);
17
18    p = p + 2;
19    printf("Adresa spre care pointează acum p este %p \n",p);
20
21    return 0;
22 }
```

Aritmetica pointerilor

- adunarea/scăderea unui număr natural dintr-un pointer

The screenshot shows a terminal window with the following content:

```
aritmeticaPointeri2.c  x
1 #include <stdio.h>
2
3 int main()
4 {
5     int v[5];
6     int *p;
7
8     p = &v[0];
9     printf("Adresa lui v[0] este %p \n", p);
10    printf("Valoarea lui v este %p \n",v);
11    printf("Adresa lui v este %p \n",&v);
12
13    if (p==v)
14        printf("Acceaasi adresa \n");
15
16    printf("%p \n %p \n" [Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc aritmeticaPointeri2.c
17                               Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
18    p =p + 2;           Adresa lui v[0] este 0x7fff51192b40
19    printf("Adresa spre [Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
20    return 0;          Valoarea lui v este 0x7fff51192b40
21
22 }                  Adresa lui v este 0x7fff51192b40
                           Aceeaasi adresa
                           0x7fff51192b40
                           0x7fff51192b44
                           0x7fff51192b48
                           0x7fff51192b4c
                           0x7fff51192b50
                           Adresa spre care pointeaza acum p este 0x7fff51192b48
```

The code demonstrates pointer arithmetic and pointer equality. It declares an integer array `v` and a pointer `p` pointing to its first element. It prints the address of `v[0]`, the value of `v`, and the address of `v`. It then adds 2 to `p` and prints the new address, showing that `p` still points to `v[0]`.

Aritmetica pointerilor

- adunarea/scăderea unui număr natural dintr-un pointer

```
aritmeticaPointeri3.c  x

01 #include <stdio.h>
02
03 int main()
04 {
05     int v[5];
06     int *p;
07
08     p = &v[0];
09     printf("Adresa spre care pointeaza acum p este %p \n",p);
10
11     p += 2;
12     printf("Adresa spre care pointeaza acum p este %p \n",p);
13
14     p++;
15     printf("Adresa spre care pointeaza acum p este %p \n",p);
16
17     p -= 3;
18     printf("Adresa spre care pointeaza acum p este %p \n",p);
19
20
21 }
```

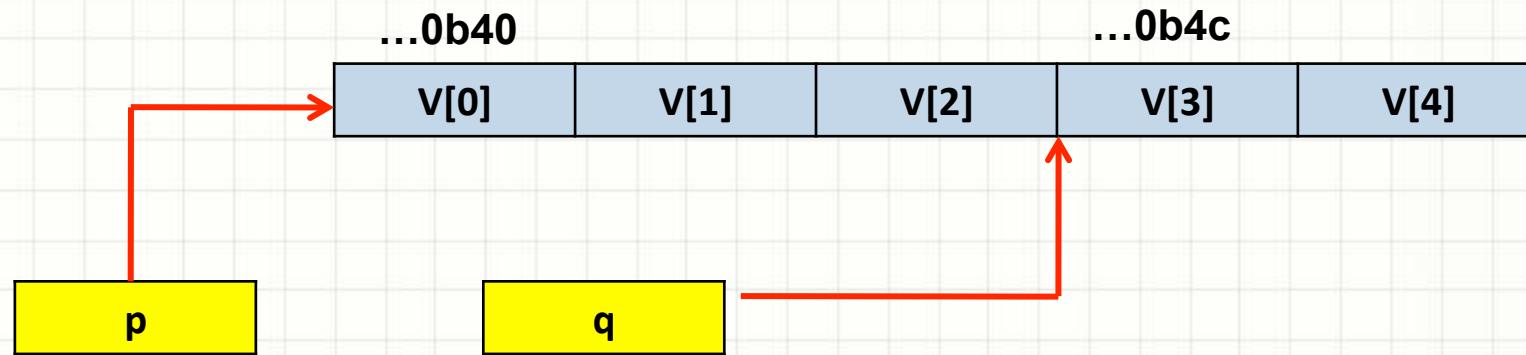
```
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc aritmeticaPointeri3.c
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
Adresa spre care pointeaza acum p este 0x7fff50f3fb40
Adresa spre care pointeaza acum p este 0x7fff50f3fb48
Adresa spre care pointeaza acum p este 0x7fff50f3fb4c
Adresa spre care pointeaza acum p este 0x7fff50f3fb40
```

Aritmetică pointerilor

- adunarea/scăderea unui număr natural dintr-un pointer
- adunarea cu n : adresa aflată peste n locații de memorie de adresa curentă stocată în pointer (“la dreapta”, se obține adăugând la adresa curentă $n * \text{sizeof}(*p)$ octeți) de același tip cu tipul de bază al variabilei de tip pointer
- scăderea cu n : adresa aflată înainte cu n locații de memorie de adresa curentă stocată în pointer (“la stânga”, se obține scăzând la adresa curentă $n * \text{sizeof}(*p)$ octeți) de același tip cu tipul de bază al variabilei de tip pointer

Aritmetica pointerilor

- scăderea a două variabile de tip pointer



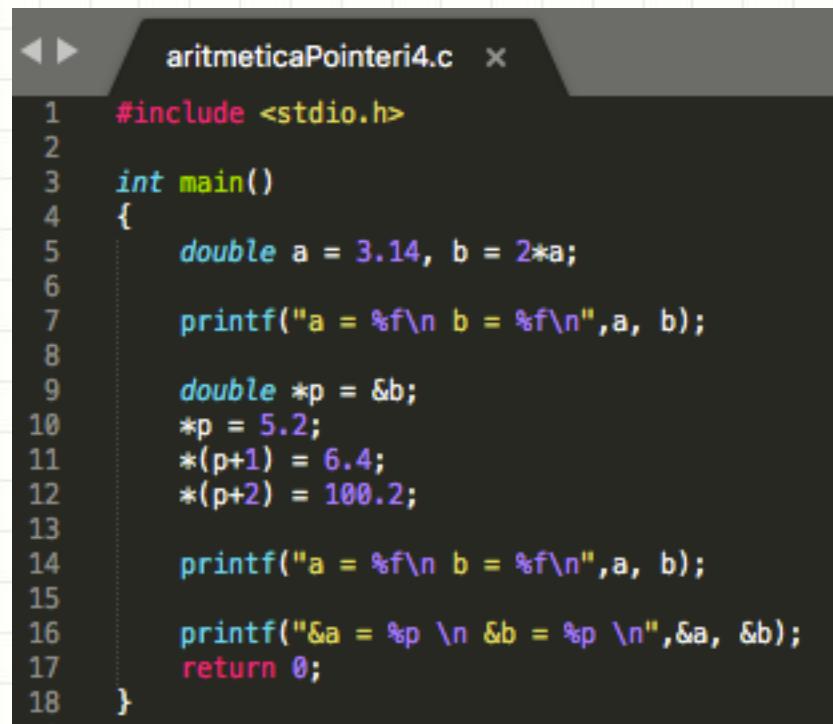
- în aritmetica pointerilor diferența dintre doi pointeri reprezintă numărul de obiecte de același tip care despart cele două adrese

Aritmetică pointerilor

- compararea a două variabile de tip pointer
 - $p - q > 0$ înseamnă că p e la dreapta lui q
 - $p - q < 0$ înseamnă că p e la stânga lui q
- compararea a două variabile de tip pointer = compararea diferenței lor cu 0

Aritmetică pointerilor

- observație: aritmetică pointerilor *are sens și este sigură* dacă adresele implicate sunt adrese ale elementelor unui tablou.



```
aritmeticaPointeri4.c  x
1 #include <stdio.h>
2
3 int main()
4 {
5     double a = 3.14, b = 2*a;
6
7     printf("a = %f\n b = %f\n",a, b);
8
9     double *p = &b;
10    *p = 5.2;
11    *(p+1) = 6.4;
12    *(p+2) = 100.2;
13
14    printf("a = %f\n b = %f\n",a, b);
15
16    printf("&a = %p \n &b = %p \n",&a, &b);
17    return 0;
18 }
```

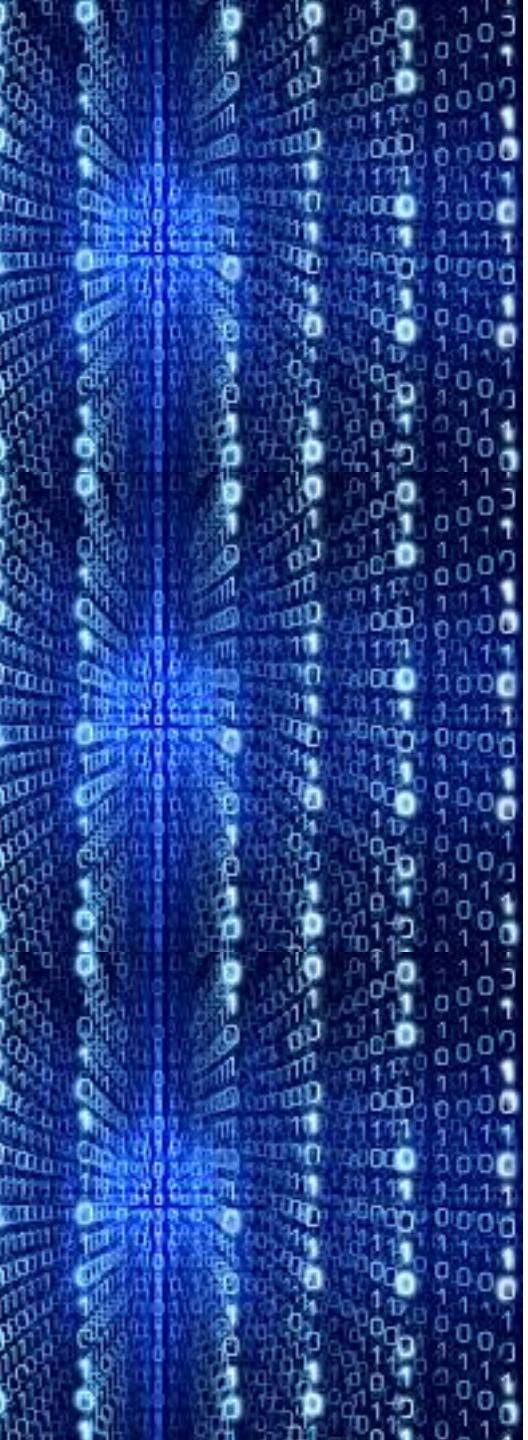
Aritmetică pointerilor

- observație: aritmetică pointerilor *are sens și este sigură* dacă adresele implicate sunt adrese ale elementelor unui tablou.

The screenshot shows a terminal window with the following content:

```
aritmeticaPointeri4.c  x
1 #include <stdio.h>
2
3 int main()
4 {
5     double a = 3.14, b = 2*a;
6
7     printf("a = %f\n b = %f\n",a, b);
8
9     double *p = &b;
10    *p = 5.2;
11    *(p+1) = 6.4;
12    *(p+2) = 100.2;
13
14    printf("a = %f\n b = %f\n",a, b);
15
16    printf("&a = %p \n &b = %p\n", &a, &b);
17    return 0;
18 }
```

[Bogdan-Alexes-MacBook-Pro:curs7 bogdan\$ gcc aritmeticaPointeri4.c
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan\$./a.out
a = 3.140000
b = 6.280000
a = 6.400000
b = 5.200000
&a = 0x7fff56c3eb50
&b = 0x7fff56c3eb48



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,

2018-2019

Cursul 8

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, siruri 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.
 - Fișiere binare**
 - Funcții specifice de manipulare.
 - Funcții (1)**
 - Declarare și definire. Apel. Metode de transmitere a parametrilor. Pointeri la funcții.
 - Tablouri și pointeri**
 - Aritmetică pointerilor
 - Legătura dintre tablouri și pointeri
 - Alocarea dinamică a memoriei
 - Clase de memorare
 - Siruri de caractere**
 - 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**
- 

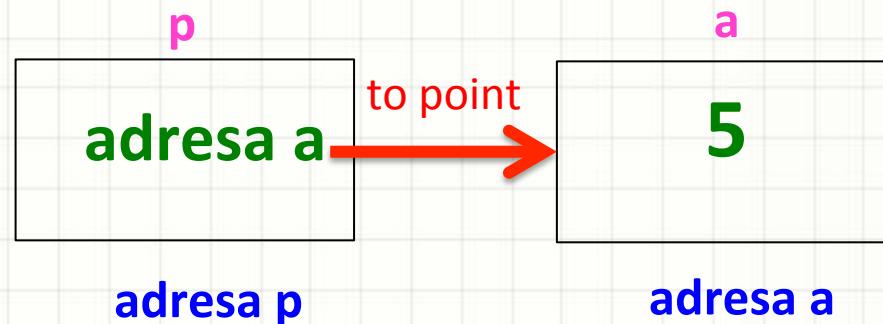
Cuprinsul cursului de azi

1. Legătura dintre tablouri și pointeri
2. Alocarea dinamică a memoriei

Legătura dintre pointeri și tablouri 1D

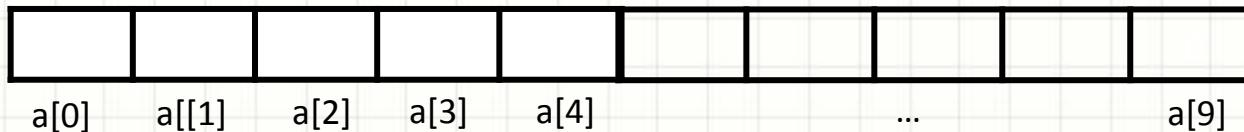
- un pointer: variabilă care poate stoca adrese de memorie

- exemplu:
int a=5
int *p;
p = &a;



- un tablou 1D: set de valori de același tip memorat la adrese succesive de memorie

- exemplu: int a[10];



Legătura dintre pointeri și tablouri 1D

- adresarea unui element dintr-un tablou cu ajutorul pointerilor
- inițializarea pointerului p cu adresa primului element al unui tablou
 - `int *p = v;`
 - `p = &v[0];`
 - **numele unui tablou este un pointer (constant) spre primul său element**

Modelatorul const

- modelatorul **const** precizează pentru o variabilă inițializată că nu este posibilă modificarea variabilei respectivă. Dacă se încearcă acest lucru se returnează eroare la compilarea programului.

```
TablouPointeri1.c
1 #include <stdio.h>
2
3 int main()
4 {
5     const int a = 10;
6     a = 5;
7
8     return 0;
9 }
```

```
Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri1.c
TablouPointeri1.c:6:4: error: cannot assign to variable 'a' with
      const-qualified type 'const int'
      a = 5;
      ^

TablouPointeri1.c:5:12: note: variable 'a' declared const here
      const int a = 10;
      ~~~~~^~~~~~
1 error generated.
```

Modelatorul const

- modelatorul **const** precizează pentru o variabilă inițializată că nu este posibilă modificarea variabilei respectivă. Dacă se încearcă acest lucru se returnează eroare la compilarea programului.
- putem modifica valoarea unei variabile însotite de modelatorul **const** prin intermediul unui pointer (în mod indirect):

```
TablouPointeri2.c      x
1 #include <stdio.h>
2
3 int main()
4 {
5     const int a = 10;
6     int *p = &a;
7
8     *p = 5;
9     printf("a = %d \n",a);
10
11
12 }
```

```
Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri2.c
TablouPointeri2.c:6:7: warning: initializing 'int *' with an
expression of type 'const int *' discards qualifiers
[-Wincompatible-pointer-types-discards-qualifiers]
    int *p = &a;
          ^ ~~~
1 warning generated.
Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
a = 10
```

Pointeri la valori constante

- modelatorul **const** poate preciza pentru un pointer că valoarea variabilei aflate la adresa conținută de pointer nu se poate modifica.

```
TablouPointeri3.c  x
1 #include <stdio.h>
2
3 int main()
4 {
5     int a = 10;
6     const int *p = &a;
7
8     *p = 5;
9     printf("a = %d \n",a);
10
11    return 0;
12 }
```

```
Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri3.c
TablouPointeri3.c:8:5: error: read-only variable is not assignable
      *p = 5;
      ~~ ^
1 error generated.
```

- putem modifica valoarea pointerului:

```
int main()
{
    int a = 10, b = 7;
    const int *p = &a;

    p = &b;
    printf("*p = %d \n", *p);

    return 0;
}
```

```
Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri4.c
Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
*p = 7
```

Pointeri constanți

- modelatorul **const** poate preciza pentru un pointer că nu poate referi o altă adresă decât cea pe care o conține la initializare.

```
2 int main()
3 {
4     int a = 10;
5     int* const p = &a;
6     printf("*p = %d \n", *p);
7
8     int b = 7;
9     p = &b;
10    printf("*p = %d \n", *p);
11    return 0;
12 }
13 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri5.c
TablouPointeri5.c:10:4: error: cannot assign to variable 'p' with
      const-qualified type 'int *const'
          p = &b;
          ^ ^
TablouPointeri5.c:6:13: note: variable 'p' declared const here
        int* const p = &a;
                    ^~~~~~
1 error generated.
```

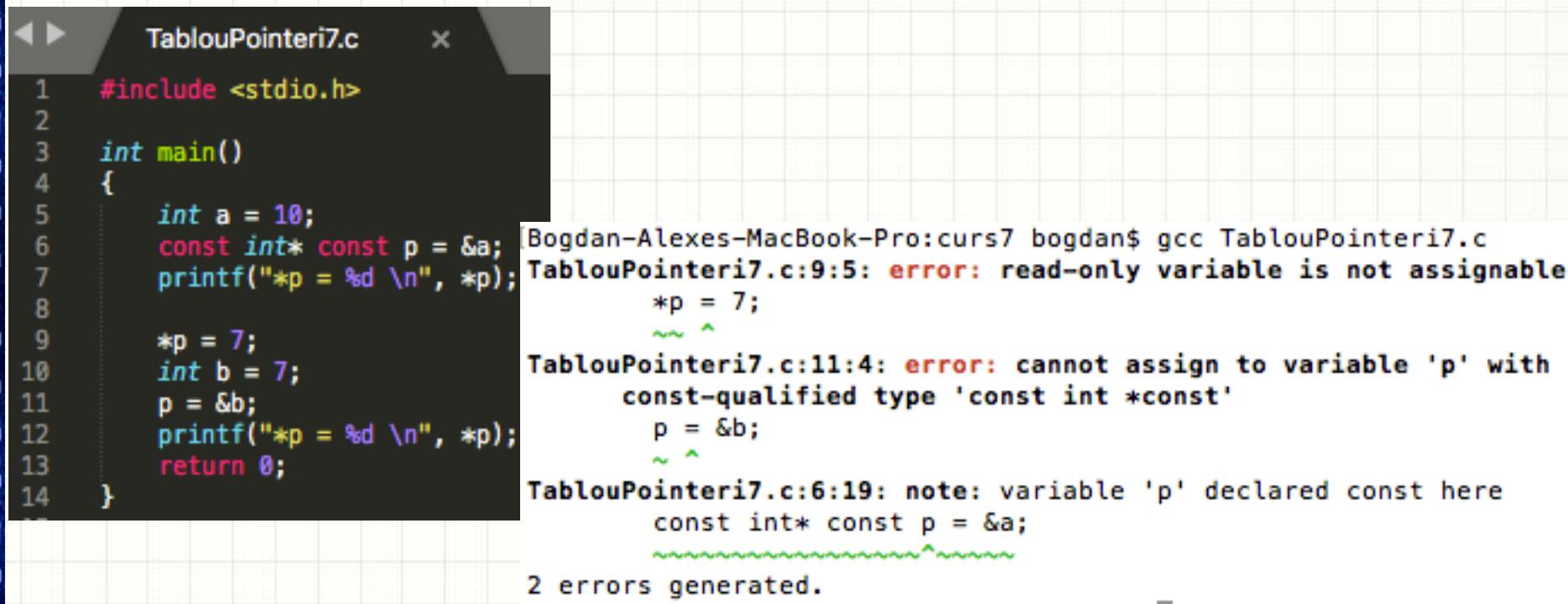
- putem modifica valoarea variabilei aflate la adresa conținută de pointer:

```
3 int main()
4 {
5     int a = 10;
6     int* const p = &a;
7     printf("*p = %d \n", *p);
8
9     *p = 7;
10    printf("*p = %d \n", *p);
11    return 0;
12 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri6.c
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
*10
*7
```

Pointeri constanți la valori constante

- modelatorul **const** poate preciza pentru un pointer că nu poate referi o altă adresă decât cea pe care o conține la initializare și de asemenea că nu poate schimba valoarea variabilei aflate la adresa pe care o conține.



```
TablouPointeri7.c      x
1 #include <stdio.h>
2
3 int main()
4 {
5     int a = 10;
6     const int* const p = &a;
7     printf("*p = %d \n", *p);
8
9     *p = 7;
10    int b = 7;
11    p = &b;
12    printf("*p = %d \n", *p);
13    return 0;
14 }
```

[Bogdan-Alexes-MacBook-Pro:curs7 bogdan\$ gcc TablouPointeri7.c
TablouPointeri7.c:9:5: error: read-only variable is not assignable
 *p = 7;
 ~~ ^
TablouPointeri7.c:11:4: error: cannot assign to variable 'p' with
 const-qualified type 'const int *const'
 p = &b;
 ~ ^
TablouPointeri7.c:6:19: note: variable 'p' declared const here
 const int* const p = &a;
~~~~~ ^~~~~~ ^~~~~~  
2 errors generated.

# Pointeri constanți vs. pointeri la valori constante

- diferențele constau în poziționarea modelatorului **const** înainte sau după caracterul \*:
  - pointer constant: int\* **const** p;
  - pointer la o constantă: **const** int\* p;
  - pointer constant la o constantă: **const** **int\*** **const** p;
- dacă declarăm o funcție astfel:

`void f(const int* p)`

atunci valorile din zona de memoria referită de p nu pot fi modificate.

# Legătura dintre pointeri și tablouri 1D

- adresarea unui element dintr-un tablou cu ajutorul pointerilor
- inițializarea pointerului p cu adresa primului element al unui tablou
  - `int *p = v;`
  - `p = &v[0];`
  - **numele unui tablou este un pointer (constant) spre primul său element**
- cum pot să găsesc adresa/valoarea celui de-al i-lea element din vectorul v pe baza pointerului p (p pointează către adresa de început a tabloului)?

# Legătura dintre pointeri și tablouri 1D

- adresarea unui element dintr-un tablou cu ajutorul pointerilor

```
TablouPointeri8.c      x
1 #include <stdio.h>
2
3 int main()
4 {
5
6     int v[5] = {0, 2, 4, 10, 20};
7     int *p = v;
8     int i;
9     for(i=0; i<5; i++)
10    {
11        printf("Accesam elementul %d din vectorul v prin intermediul lui p.\n",i);
12        printf("%d %d\n", *(p+i),p[i]);
13    }
14    return 0;
15 }
```

Bogdan-Alexes-MacBook-Pro:curs8 bogdan\$ gcc TablouPointeri8.c  
Bogdan-Alexes-MacBook-Pro:curs8 bogdan\$ ./a.out  
Accesam elementul 0 din vectorul v prin intermediul lui p.  
0 0  
Accesam elementul 1 din vectorul v prin intermediul lui p.  
2 2  
Accesam elementul 2 din vectorul v prin intermediul lui p.  
4 4  
Accesam elementul 3 din vectorul v prin intermediul lui p.  
10 10  
Accesam elementul 4 din vectorul v prin intermediul lui p.  
20 20

# Legătura dintre pointeri și tablouri 1D

- adresarea unui element dintr-un tablou cu ajutorul pointerilor
- adresa lui  $v[i]$ :  $\&v[i] = \&p[i] = p + i$
- valoarea lui  $v[i]$ :  $v[i] = p[i] = *(p + i)$
- comutativitate:  $v[i] = *(p + i) = *(i + p) = i[v] ?!$

# Legătura dintre pointeri și tablouri 1D

- adresarea unui element dintr-un tablou cu ajutorul pointerilor

```
TablouPointeri9.c      x
1 #include <stdio.h>
2
3 int main()
4 {
5
6     int v[5] = {0, 2, 4, 10, 20};
7     int i;
8
9     printf("Afisare cu v[i] \n");
10    for(i=0; i<5; i++)
11        printf("v[%d] = %d \n", i, v[i]);
12
13    printf("Afisare cu i[v] \n");
14    for(i=0; i<5; i++)
15        printf("%d[v] = %d \n", i, i[v]);
16
17    return 0;
18 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ gcc TablouPointeri9.c
[Bogdan-Alexes-MacBook-Pro:curs7 bogdan$ ./a.out
Afisare cu v[i]
v[0] = 0
v[1] = 2
v[2] = 4
v[3] = 10
v[4] = 20
Afisare cu i[v]
0[v] = 0
1[v] = 2
2[v] = 4
3[v] = 10
4[v] = 20
```

# Legătura dintre pointeri și tablouri 1D

- adresarea unui element dintr-un tablou cu ajutorul pointerilor
- numele unui tablou este un pointer (**constant**) spre primul său element.
- la compilare, expresia  $v[i]$  se înlocuiește cu  $*(v+i)$ . Atunci,  **$i[v]$  este o expresie corectă** întrucât  $i[v]$  se înlocuiește cu  $*(i+v) = *(v+i)$ . Deci,  $v[i] = i[v]$ .
- $\&v[i] = \&(*(v+i)) = v + i \Rightarrow \&v[0] = v$

# Legătura dintre pointeri și tablouri 1D

- numele unui tablou este un pointer constant spre primul său element.

`int v[100];`   $v = \&v[0];$

`int a[10][10];`   $a = \&a[0][0];$

- elementele unui tablou pot fi accesate prin pointeri:

| index              | 0      | 1        | i        | n-1        |
|--------------------|--------|----------|----------|------------|
| accesare directă   | $v[0]$ | $v[1]$   | $v[i]$   | $v[n-1]$   |
| accesare indirectă | $*v$   | $*(v+1)$ | $*(v+i)$ | $*(v+n-1)$ |
| adresa             | $v$    | $v+1$    | $v+i$    | $v+n-1$    |

- operatorul \* are prioritate mai mare ca +
- $*(v+1)$  e diferit de  $*v+1$

# Legătura dintre pointeri și tablouri 1D

| index              | 0    | 1      | i      | n-1      |
|--------------------|------|--------|--------|----------|
| accesare directă   | v[0] | v[1]   | v[i]   | v[n-1]   |
| accesare indirectă | *v   | *(v+1) | *(v+i) | *(v+n-1) |
| adresa             | v    | v+1    | v+i    | v+n-1    |

- o expresie cu tablou și indice este echivalentă cu una scrisă ca pointer și distanță de deplasare:  $v[i] = *(v+i)$

# Diferențe între pointeri și tablouri 1D

- un pointer își poate schimba valoarea:  $p = v$  și  $p++$  **sunt expresii corecte**
- un nume de tablou este un pointer constant (nu își poate schimba valoarea):  $v = p$  și  $v++$  **sunt expresii incorecte**
- `sizeof(v)` și `sizeof(p)` sunt de obicei diferite  
`int v[10];`  
`int *p = v;`  
`sizeof(v) -> 40 de octeți`  
`sizeof(p) -> 8 octeți`

# Legătura dintre pointeri și tablouri 2D

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

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

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



|         |         |         |         |         |         |     |   |    |    |    |    |   |   |         |
|---------|---------|---------|---------|---------|---------|-----|---|----|----|----|----|---|---|---------|
| 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

- ❑ **tablou bidimensional = tablou de tablouri**

|   |     |    |   |   |
|---|-----|----|---|---|
| 3 | -12 | 10 | 7 | 1 |
|---|-----|----|---|---|

a[0]

|    |   |   |    |    |
|----|---|---|----|----|
| 10 | 2 | 0 | -7 | 41 |
|----|---|---|----|----|

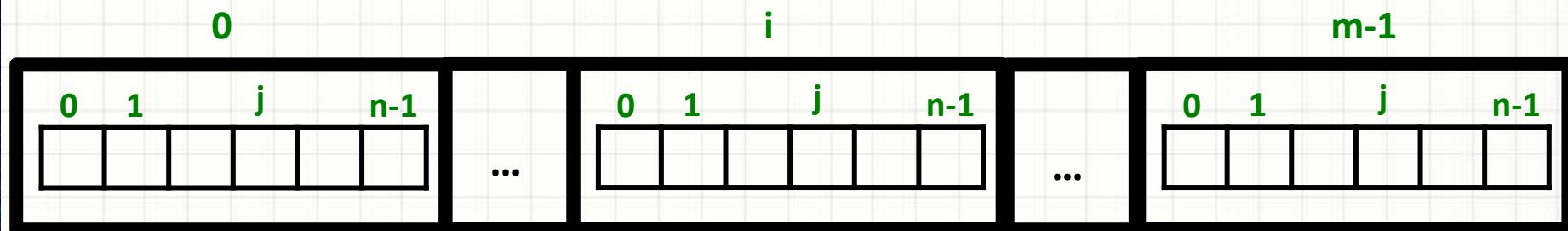
a[1]

|    |    |   |   |   |
|----|----|---|---|---|
| -3 | -2 | 0 | 0 | 2 |
|----|----|---|---|---|

a[2]

# Legătura dintre pointeri și tablouri 2D

- **tablou bidimensional = tablou de tablouri**
- **cazul general:** int a[m][n];

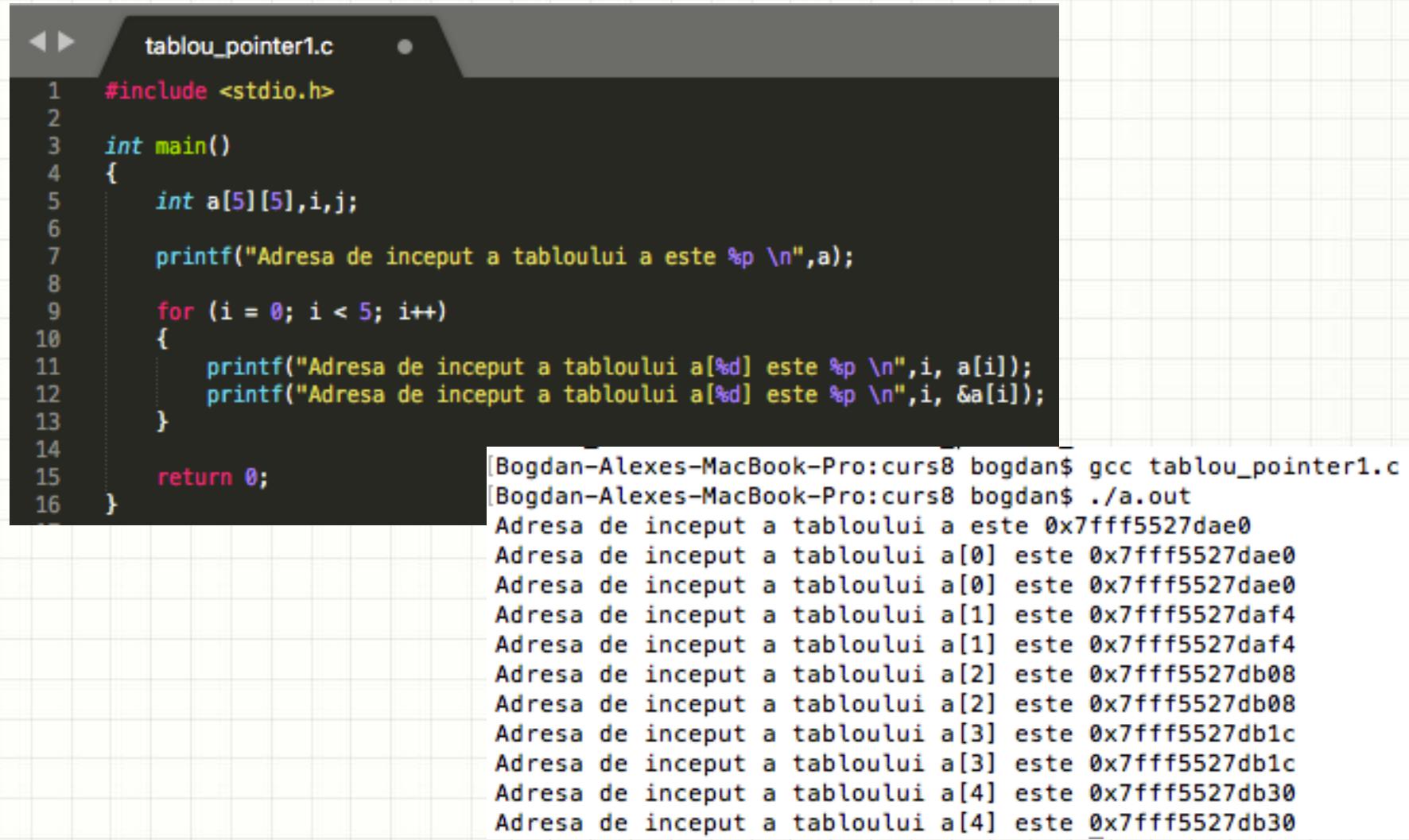


Reprezentarea în memoria calculatorului a unui tablou bidimensional

- a este tablou bidimensional;
- elementele lui a sunt tablouri unidimensionale care ocupă  $\text{sizeof}(\text{int}) * n = 4 * n$  octeți.
- elementele lui a:  $a[0] = *(a+0)$ ,  $a[1] = *(a+1)$ , ...,  $a[m-1] = *(a+m-1)$
- **tabloul  $a[i]$  începe la adresa  $a+i$  (=  $a+i*4*n$  octeți în aritmetica pointerilor)**

# Legătura dintre pointeri și tablouri 2D

## □ tablou bidimensional = tablou de tablouri

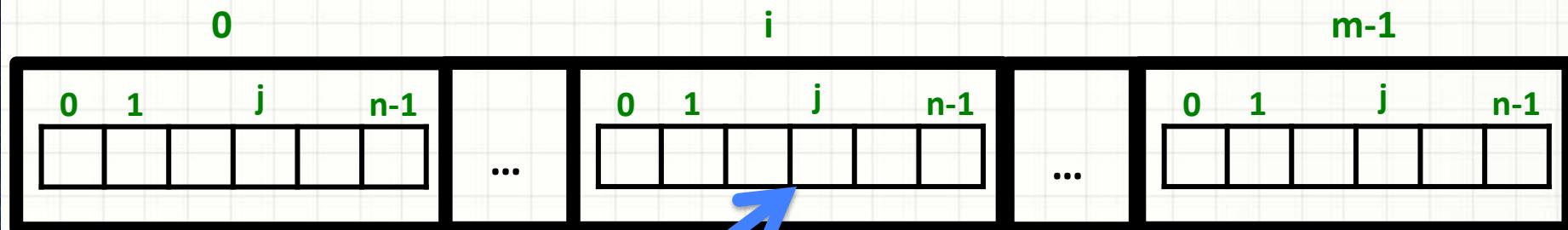


```
tablou_pointer1.c
1 #include <stdio.h>
2
3 int main()
4 {
5     int a[5][5],i,j;
6
7     printf("Adresa de inceput a tabloului a este %p \n",a);
8
9     for (i = 0; i < 5; i++)
10    {
11        printf("Adresa de inceput a tabloului a[%d] este %p \n",i, a[i]);
12        printf("Adresa de inceput a tabloului a[%d] este %p \n",i, &a[i]);
13    }
14
15    return 0;
16 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc tablou_pointer1.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Adresa de inceput a tabloului a este 0x7fff5527dae0
Adresa de inceput a tabloului a[0] este 0x7fff5527dae0
Adresa de inceput a tabloului a[0] este 0x7fff5527dae0
Adresa de inceput a tabloului a[1] este 0x7fff5527daf4
Adresa de inceput a tabloului a[1] este 0x7fff5527daf4
Adresa de inceput a tabloului a[2] este 0x7fff5527db08
Adresa de inceput a tabloului a[2] este 0x7fff5527db08
Adresa de inceput a tabloului a[3] este 0x7fff5527db1c
Adresa de inceput a tabloului a[3] este 0x7fff5527db1c
Adresa de inceput a tabloului a[4] este 0x7fff5527db30
Adresa de inceput a tabloului a[4] este 0x7fff5527db30
```

# Legătura dintre pointeri și tablouri 2D

- **tablou bidimensional = tablou de tablouri**
- **cazul general:** int a[m][n];



Reprezentarea în memoria calculatorului a unui tablou bidimensional

- tabloul  $a[i]$  începe la adresa  $a+i$  ( $= a+i*4*n$  octeți în aritmetică pointerilor)
- care este **adresa** lui  $a[i][j]$ ? **Cum o exprim în aritmetică pointerilor în funcție de  $a$ ,  $i$ ,  $j$ ?**
- **adresa lui  $a[i][j] = *(a+i)+j$**

# Legătura dintre pointeri și tablouri 2D

## □ tablou bidimensional = tablou de tablouri

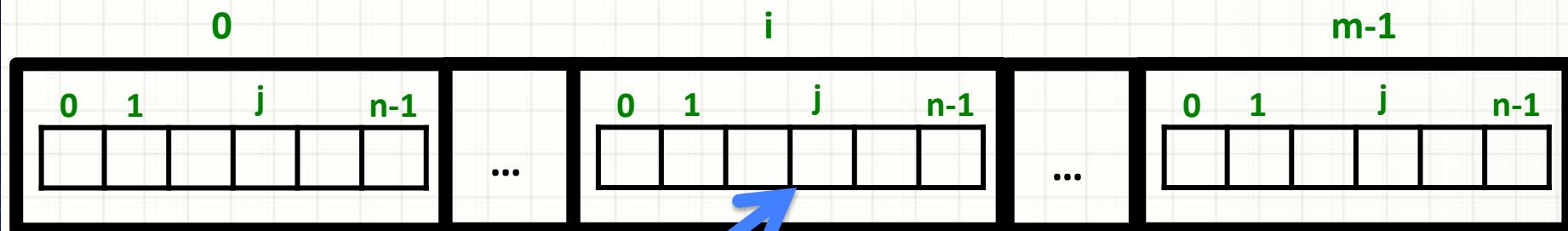
```
tablou_pointer2.c  x

1 #include <stdio.h>
2
3 int main()
4 {
5     int a[5][5],i,j;
6
7     i = 3;
8
9     for (j = 0; j < 5; j++)
10    {
11        printf("Adresa lui a[%d][%d] este %p \n",i, j, &a[i][j]);
12        printf("Adresa lui a[%d][%d] este %p \n",i, j, *(a+i)+j);
13    }
14
15    return 0;
16 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc tablou_pointer2.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Adresa lui a[3][0] este 0x7fff512d0b1c
Adresa lui a[3][0] este 0x7fff512d0b1c
Adresa lui a[3][1] este 0x7fff512d0b20
Adresa lui a[3][1] este 0x7fff512d0b20
Adresa lui a[3][2] este 0x7fff512d0b24
Adresa lui a[3][2] este 0x7fff512d0b24
Adresa lui a[3][3] este 0x7fff512d0b28
Adresa lui a[3][3] este 0x7fff512d0b28
Adresa lui a[3][4] este 0x7fff512d0b2c
Adresa lui a[3][4] este 0x7fff512d0b2c
```

# Legătura dintre pointeri și tablouri 2D

- **tablou bidimensional = tablou de tablouri**
- **cazul general:** int a[m][n];



Reprezentarea în memoria calculatorului a unui tablou bidimensional

- tabloul  $a[i]$  începe la adresa  $a+i$  ( $= a+i*4*n$  octeți în aritmetică pointerilor)
- care este **adresa** lui  $a[i][j]$ ? **Cum o exprim în aritmetică pointerilor în funcție de a, i, j?** **adresa lui  $a[i][j] = *(a+i)+j$**
- cum exprim **valoarea  $a[i][j]$  în aritmetică pointerilor în funcție de a, i, j?**
- $a[i][j] = *(*(a+i)+j)$  ( $a$  este pointer dublu)

# Legătura dintre pointeri și tablouri 2D

## □ tablou bidimensional = tablou de tablouri

```
tablou_pointer3.c      x
1 #include <stdio.h>
2
3 int main()
4 {
5     int a[5][5],i,j;
6
7     for(i = 0; i < 5; i++)
8         for(j = 0; j < 5; j++)
9             a[i][j] = i*j;
10
11    i = 3;
12
13    for (j=0;j<5;j++)
14    {
15        printf("Valoarea lui a[%d][%d] este %d \n",i, j, a[i][j]);
16        printf("Valoarea lui a[%d][%d] este %d \n",i, j, *(*(a+i)+j));
17    }
18
19    return 0;
20 }
```

```
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc tablou_pointer3.c
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Valoarea lui a[3][0] este 0
Valoarea lui a[3][0] este 0
Valoarea lui a[3][1] este 3
Valoarea lui a[3][1] este 3
Valoarea lui a[3][2] este 6
Valoarea lui a[3][2] este 6
Valoarea lui a[3][3] este 9
Valoarea lui a[3][3] este 9
Valoarea lui a[3][4] este 12
Valoarea lui a[3][4] este 12
```

# Legătura dintre pointeri și tablouri 2D

- **tablou bidimensional = tablou de tablouri**
- **cazul general:** int a[m][n];
- **adresa lui  $a[i][j] = *(a+i)+j$**
- **valoarea lui  $a[i][j] = *(*(a+i)+j)$**
- **știu că  $a[i] = *(a+i) = i[a]$ . Atunci  $a[i][j]$  se mai poate scrie ca:**
  - $*(a[i]+j)$
  - $*(i[a] + j)$
  - $(*(a+i))[j]$
  - $i[a][j]$
  - $j[i[a]]$
  - $j[a[i]]$

# Trasmiterea tablourilor ca argumente funcțiilor

- pentru tablouri 1D, se transmite adresa primului element + lungimea tabloului (nu am cum sa o iau de altundeva)

```
int numeFunctie(int v[], int n)
```

```
int numeFunctie(int v[10], int n)
```

```
int numeFunctie(int* v, int n)
```



sunt echivalente

```
transmitereTablou.c  x
1 #include <stdio.h>
2
3 void afiseazaTablou(int v[])
4 {
5     int i;
6     for(i=0;i<sizeof(v)/sizeof(int);i++)
7         printf("%d \n", v[i]);
8
9     printf("Dimensiunea lui v este %lu \n", sizeof(v));
10 }
11
12 int main()
13 {
14     int v[5] = {1,3,5,7,9};
15     afiseazaTablou(v);
16     return 0;
17 }
```

```
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
1
3
Dimensiunea lui v este 8
5
7
```

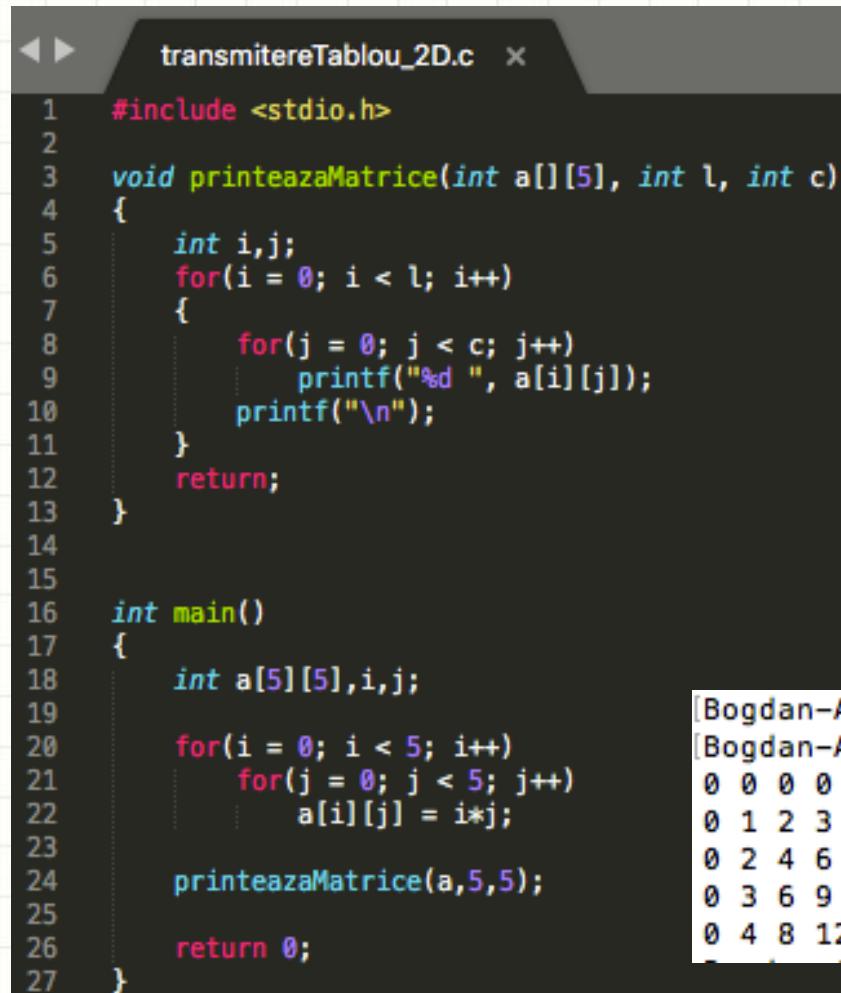
Nu afiseaza ceea ce vreau!!!  
(tabloul e vazut ca un pointer  
in functie) -> tabloul  
“decade” la un pointer

# Trasmiterea tablourilor ca argumente funcțiilor

- ❑ pentru tablouri 2D, trebuie să transmit neapărat a doua dimensiune (compilatorul trebuie să știe câte elemente are o "linie")
- ❑ pe cazul general nD trebuie să transmit toate dimensiunile (prima poate lipsi)

# Trasmiterea tablourilor ca argumente funcțiilor

- pentru tablouri 2D, trebuie să transmit neapărat a doua dimensiune (compilatorul trebuie să știe câte elemente are o "linie")



```
transmitereTablou_2D.c  x

1 #include <stdio.h>
2
3 void printeazaMatrice(int a[][5], int l, int c)
4 {
5     int i,j;
6     for(i = 0; i < l; i++)
7     {
8         for(j = 0; j < c; j++)
9             printf("%d ", a[i][j]);
10        printf("\n");
11    }
12    return;
13 }

14
15
16 int main()
17 {
18     int a[5][5],i,j;
19
20     for(i = 0; i < 5; i++)
21         for(j = 0; j < 5; j++)
22             a[i][j] = i*j;
23
24     printeazaMatrice(a,5,5);
25
26     return 0;
27 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc transmitereTablou_2D.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
0 0 0 0 0
0 1 2 3 4
0 2 4 6 8
0 3 6 9 12
0 4 8 12 16]
```

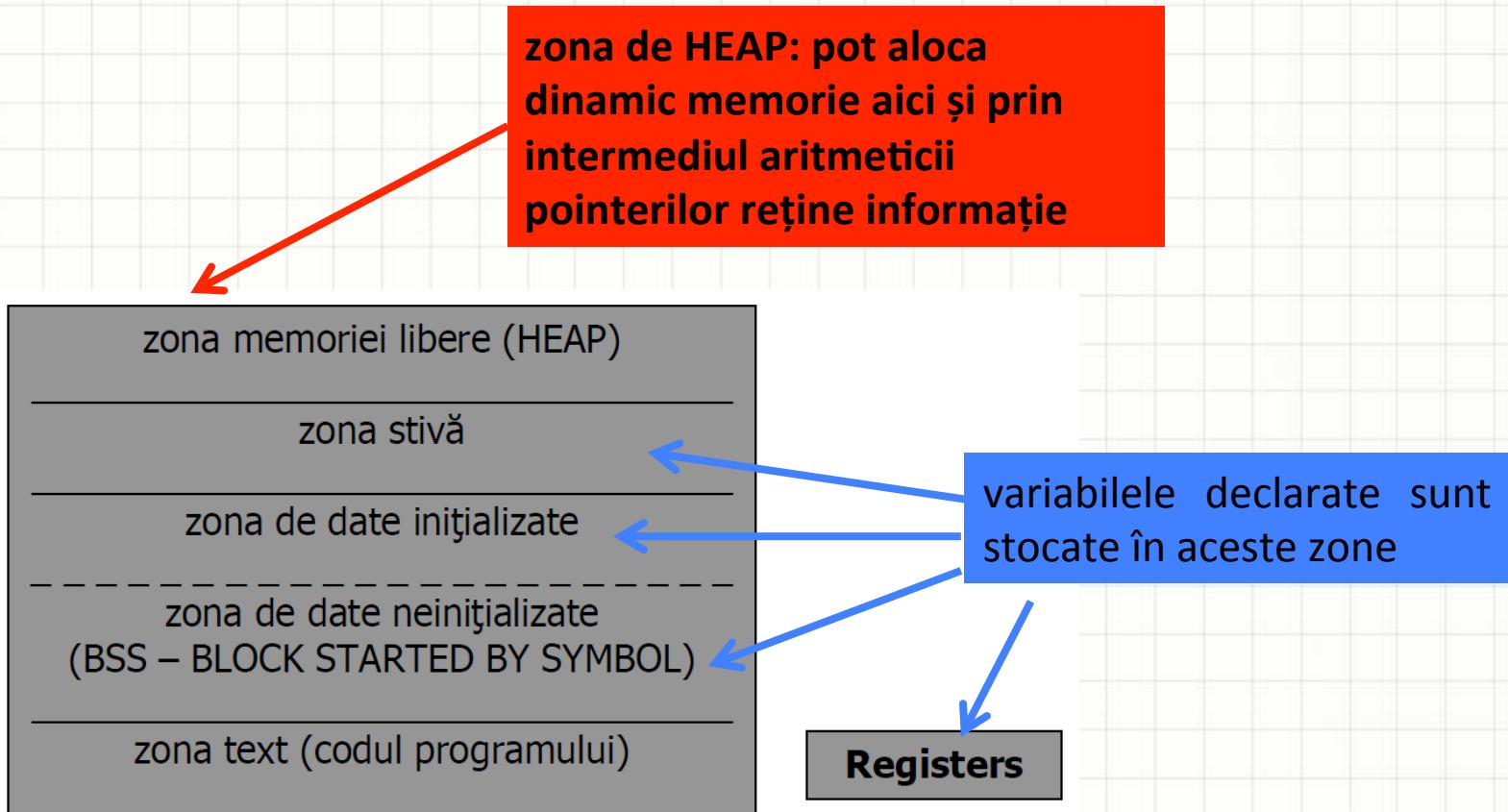
# Cuprinsul cursului de azi

1. Legătura dintre tablouri și pointeri.
2. Alocarea dinamică a memoriei.

# Alocarea dinamică a memoriei

- de cele mai multe ori lucrăm într-un program cu tablouri a căror dimensiune nu o cunoaștem înainte de execuție (o putem aproxima, de obicei alocăm mai multă memorie la compilare).
- în alocarea statică (`int a[50];`) sistemul intern de alocare a memoriei alocă memorie pentru tablouri în zona de STACK (stivă). Tablourile alocate static au durată de viață egală cu timpul de execuție al programului (nu se pot dezaloca).
- dimensiunea zonei de STACK e limitată (implicit câțiva MB), nu putem aloca tablouri foarte mari;
- alocarea dinamică ne permite gestionarea eficientă a memoriei:
  - alocăm memorie exact cât avem nevoie
  - putem să eliberăm memoria folosită la un moment dat

# Harta simplificată a memoriei la rularea unui program



# Alocarea dinamică a memoriei

- *heap*-ul este o zonă predefinită de memorie (de dimensiuni foarte mari) care poate fi accesată de program pentru a stoca date și variabile;
- datele și variabilele pot fi alocate pe *heap* prin apeluri speciale de funcții din biblioteca *stdlib.h*: **malloc**, **calloc**, **realloc**
- zonele de memorie pot să fie dezalocate la cerere prin apelul funcției **free**
- este recomandat ca memoria să fie eliberată în momentul în care datele/variabilele respective nu mai sunt de interes!

# Funcția malloc

- ❑ prototipul funcției:

**void \* malloc( int dimensiune);**

unde:

- ❑ **dimensiune** = numărul de octeți ceruți a se aloca
- ❑ dacă există suficient spațiu liber în HEAP atunci un bloc de memorie continuu de dimensiunea specificată va fi marcat ca ocupat, iar **funcția malloc va returna un pointer ce conține adresa de început a acelui bloc.** Dacă nu există suficient spațiu liber funcția malloc întoarce **NULL** ( pointer de tip void\* la adresa de memorie 0 – adresa nevalidă, nu putem stoca date acolo).
- ❑ accesarea blocului alocat se realizează printr-un pointer (**din STACK**) către adresa de început a blocului (**din HEAP**).

# Funcția malloc

- ❑ prototipul funcției:

**void \* malloc( int dimensiune);**

unde:

- ❑ **dimensiune** = numărul de octeți ceruți a se aloca
- ❑ tipul generic **void \*** returnat de funcția malloc face obligatorie utilizarea unei conversii de tip atunci când respectivul pointer este asignat unui pointer de tip obișnuit.
- ❑ pointerul în care păstrăm adresa returnată de malloc va fi plasat în zona de memorie statică.

# Functia malloc

## exemplu:

```
alocareDinamica1.c  x

01 #include <stdio.h>
02 #include <stdlib.h>
03
04 int main()
05 {
06
07     int a = 0;
08     int *p = &a;
09     printf("Adresa lui a este = %p \n", &a);
10     printf("Adresa lui p este = %p \n", &p);
11     printf("Cerere alocare memorie in HEAP\n");
12     p = (int *) malloc(5*sizeof(int));
13     if (p == NULL)
14     {
15         printf("Nu exista spatiu liber in HEAP \n");
16         return 1;
17     }
18     printf("Pointerul p pointeaza catre adresa = %p din HEAP\n",p);
19     for (int i = 0; i < 5; i++)
20     {
21         p[i] = i;
22     }
23     free(p);
24
25 }
```

[Bogdan-Alexes-MacBook-Pro:curs8 bogdan\$ gcc alocareDinamica1.c  
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan\$ ./a.out  
Adresa lui a este = 0x7fff5706fb48  
Adresa lui p este = 0x7fff5706fb40  
Cerere alocare memorie in HEAP  
Pointerul p pointeaza catre adresa = 0x7fb06c025b0 din HEAP

# Funcția malloc

## □ exemplu:

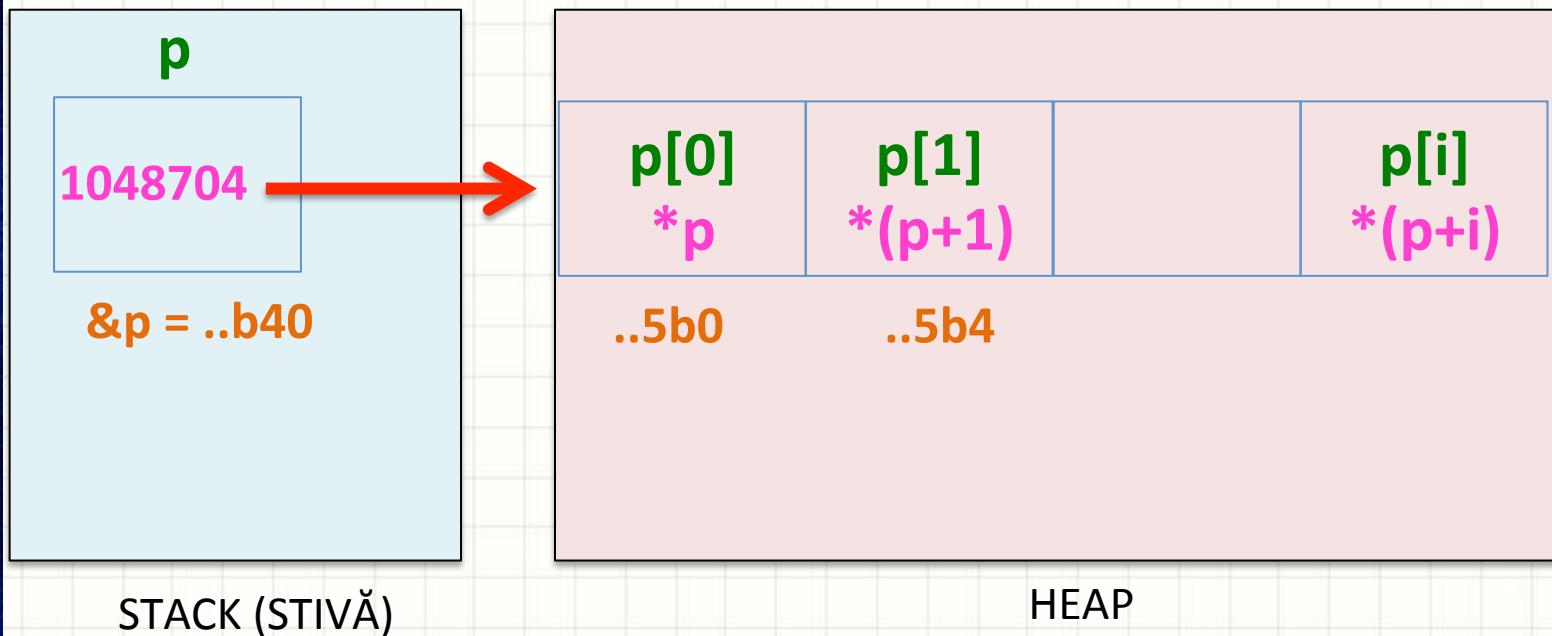
```
alocareDinamica1.c  x
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6
7     int a = 0;
8     int *p = &a;
9     printf("Adresa lui a este = %p \n", &a);
10    printf("Adresa lui p este = %p \n", &p);
11    printf("Cerere alocare memorie in HEAP\n");
12    p = (int *) malloc(5*sizeof(int));
13    if (p == NULL)
14    {
15        printf("Nu exista spatiu liber in HEAP \n");
16        return 1;
17    }
18    printf("Pointerul p pointeaza catre adresa = %p \n");
19    for (int i = 0; i < 5; i++)
20    {
21        p[i] = i;
22    }
23    free(p);
24
25    return 0;
26 }
```

## Observatie

- blocurile alocate în zona de memorie dinamică nu au nume → mod de acces: adresa de memorie.
- accesul blocului de memorie se realizează prin intermediul unui pointer în care păstrăm adresa de început.
- orice bloc de memorie alocat dinamic trebuie *elibерат* înainte să se încheie execuția programului. Funcția **free** permite eliberarea memoriei (parametru: adresa de început a blocului).

# Functia malloc

## □ exemplu:



```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc alocareDinamica1.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Adresa lui a este = 0xffff5706fb48
Adresa lui p este = 0xffff5706fb40
Cerere alocare memorie in HEAP
Pointerul p pointeaza catre adresa = 0x7fb06c025b0 din HEAP
```

# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.

```
alocareDinamica2.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int citire(int* v)
5 {
6     int i,n;
7     printf("n=");scanf("%d",&n);
8     v = (int *) malloc(n*sizeof(int));
9     for(i = 0; i < n; i++)
10        scanf("%d", &v[i]);
11     return n;
12 }
13
14 int main()
15 {
16     int n, i, *p = NULL;
17     n = citire(p);
18     for(i = 0; i < n; i++)
19         printf("p[%d] = %d\n",i,p[i]);
20
21     return 0;
22 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc alocareDinamica2.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
n=5
10
20
30
40
50
Segmentation fault: 11
```

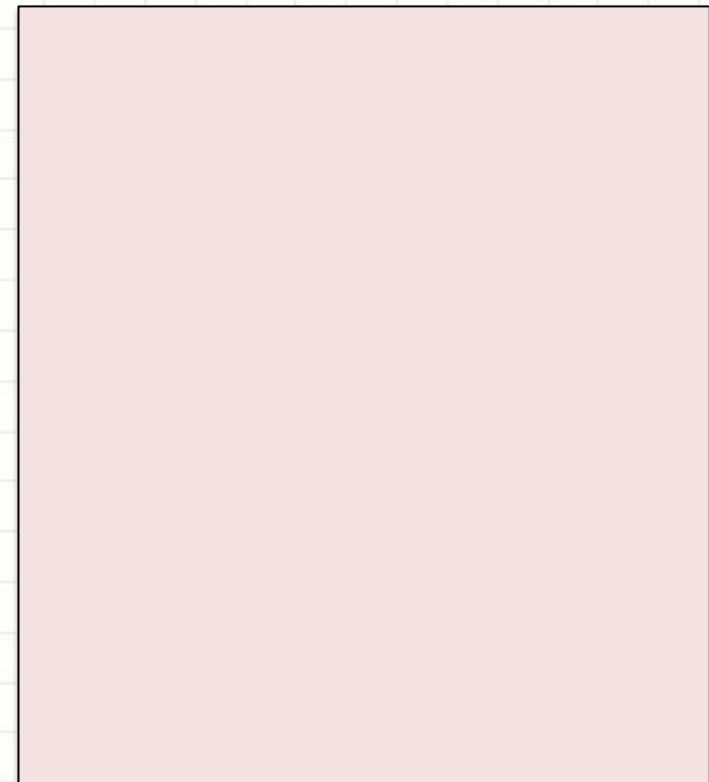
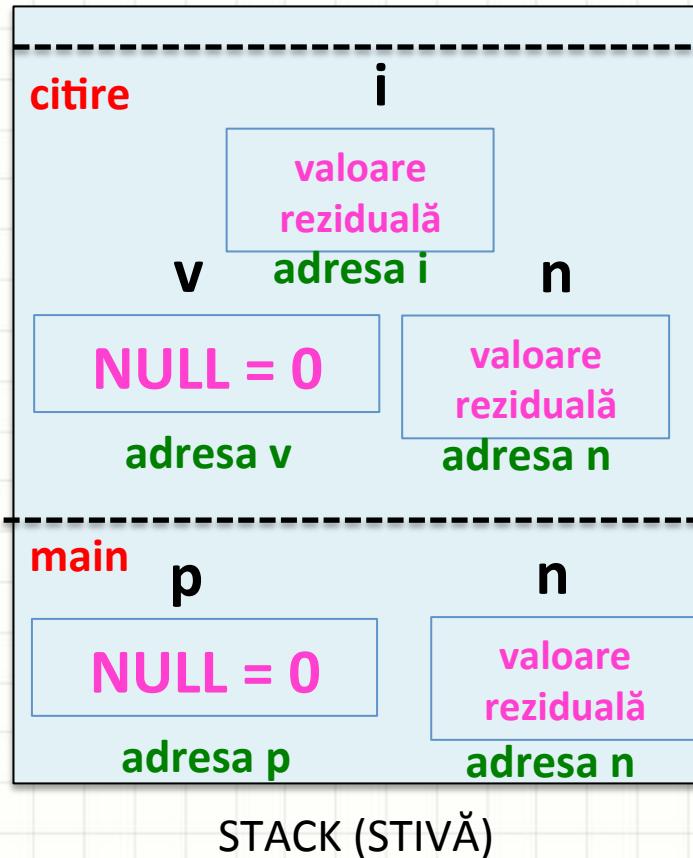
De ce nu se afișează vectorul citit?

Transmiterea unui pointer nu  
înseamnă simularea transmiterii  
prin referință

# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.

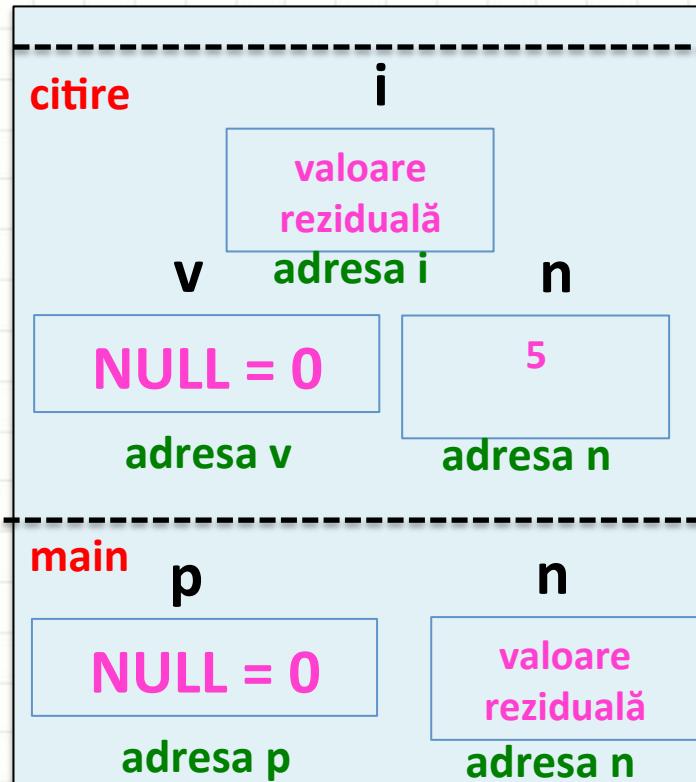
v este  
copie a  
lui p



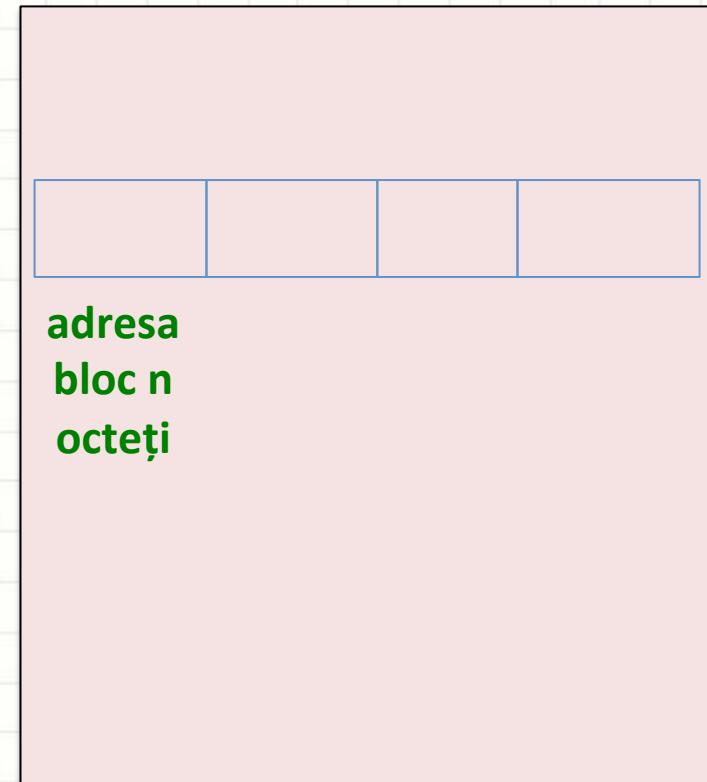
# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.

v este  
copie a  
lui p



STACK (STIVĂ)

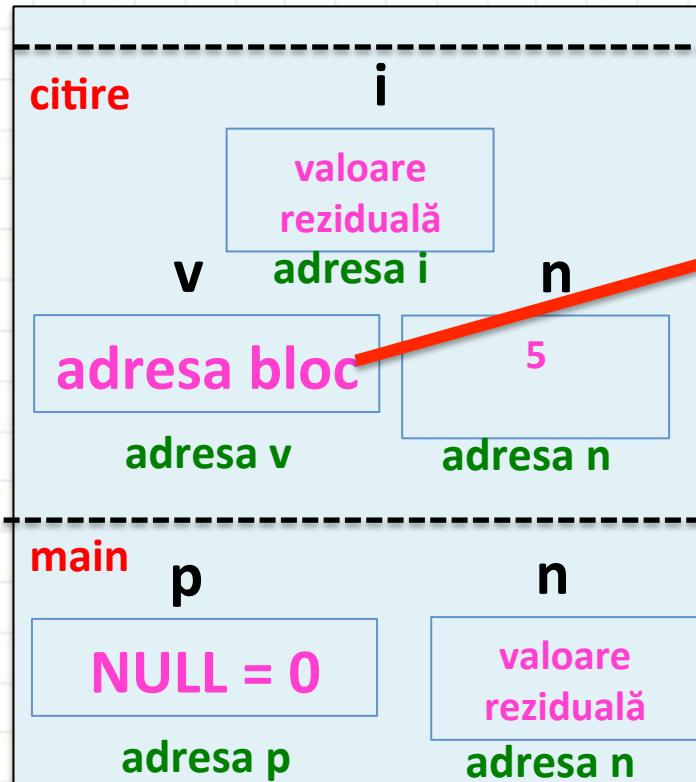


HEAP

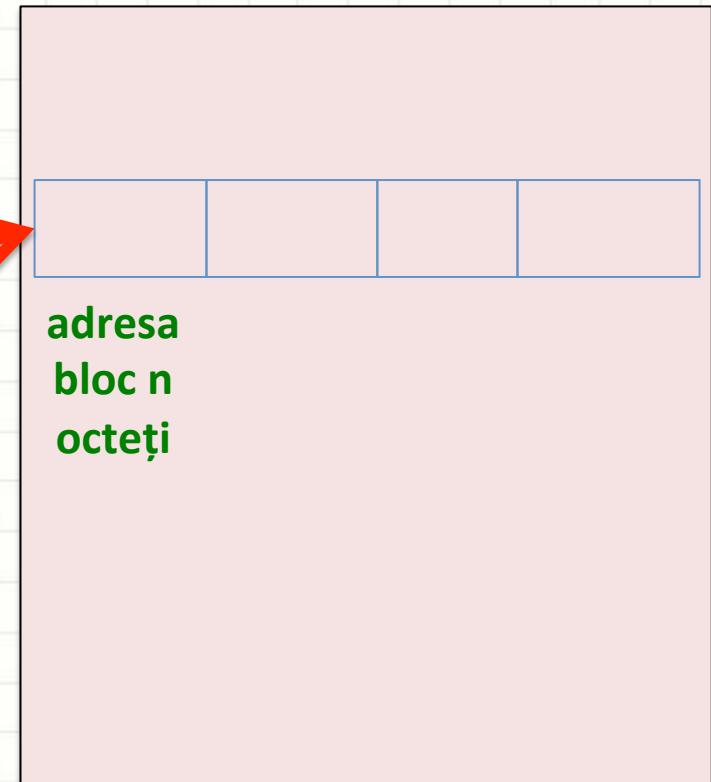
# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.

v este  
copie a  
lui p



STACK (STIVĂ)

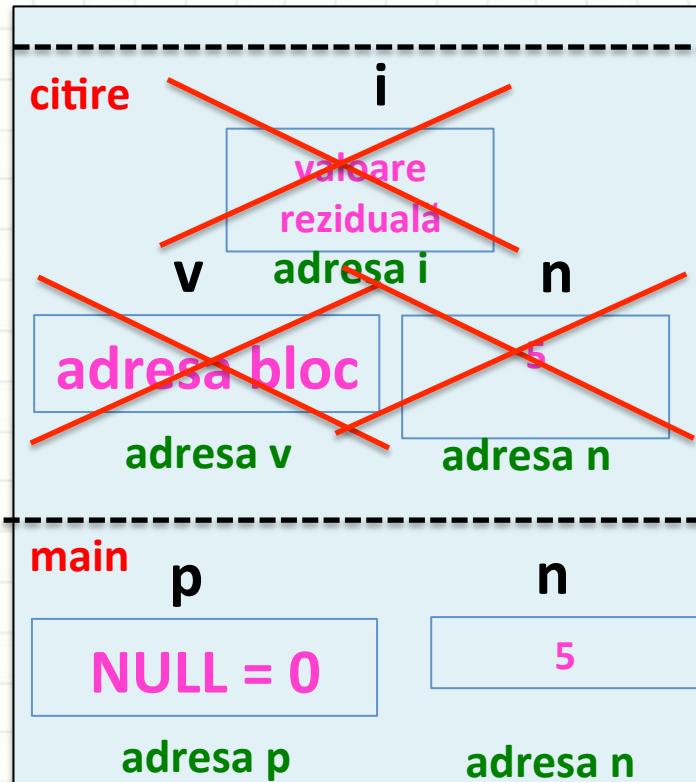


HEAP

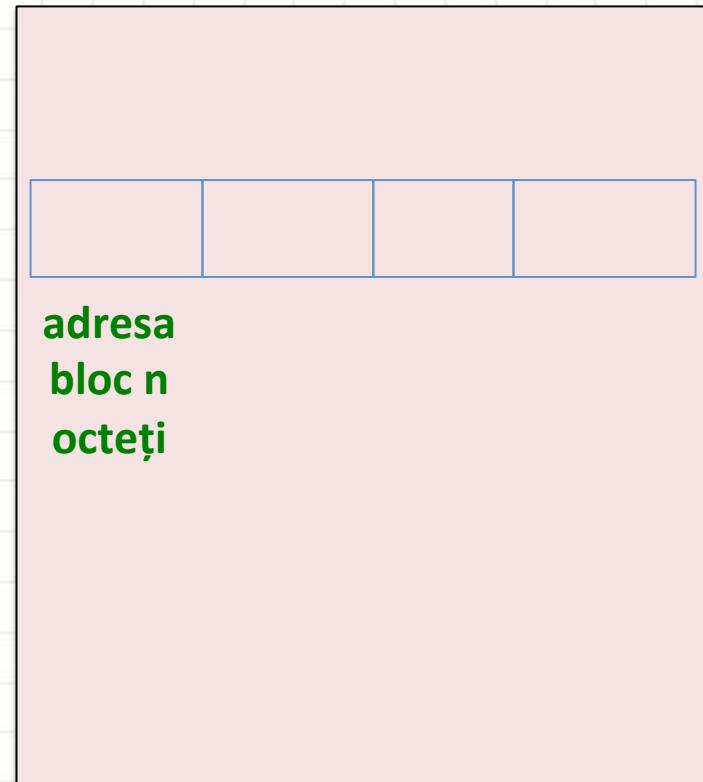
# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.

v se  
distrugă,  
se  
întoarce  
5



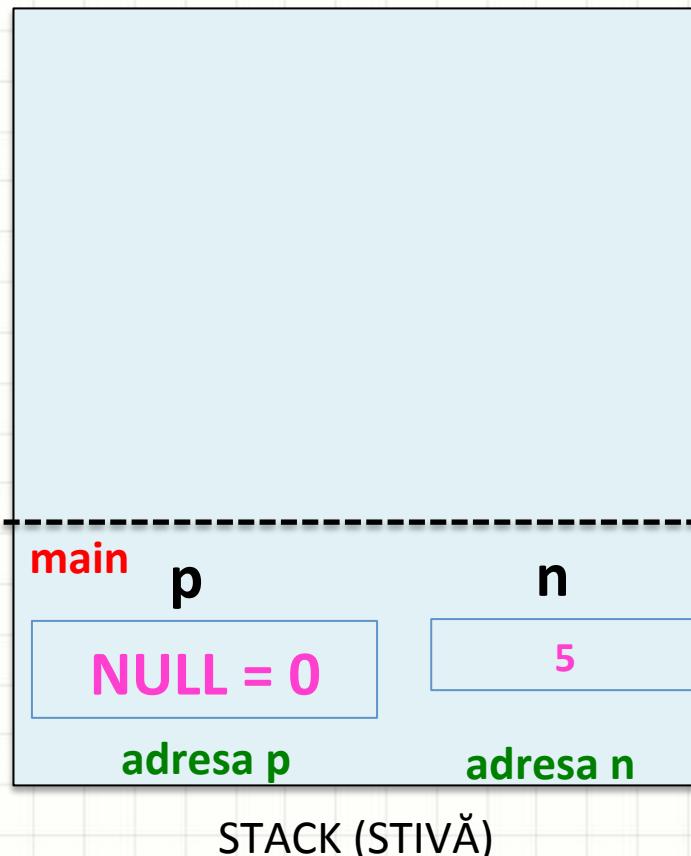
STACK (STIVĂ)



HEAP

# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.



# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.

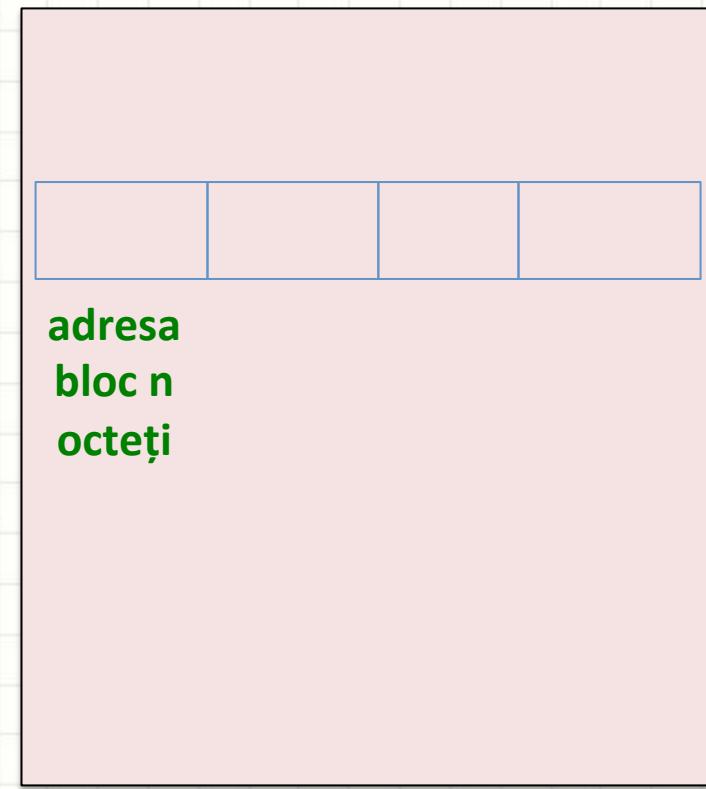
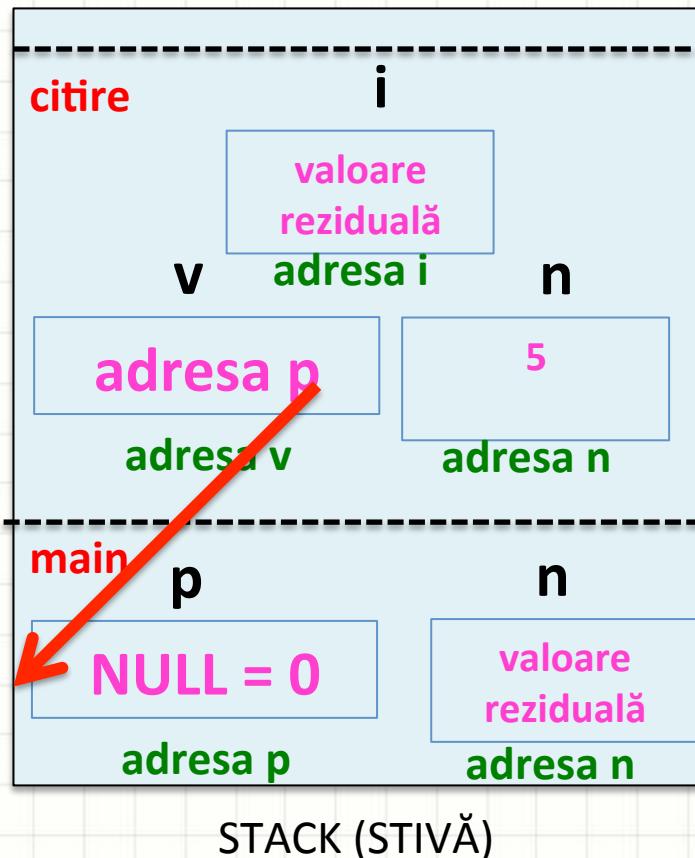
```
alocareDinamica3.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int citire(int** v)
5 {
6     int i,n;
7     printf("n=");scanf("%d",&n);
8     *v = (int *) malloc(n*sizeof(int));
9     for(i = 0; i < n; i++)
10        scanf("%d", &(*v)[i]);
11     return n;
12 }
13
14 int main()
15 {
16     int n, i, *p = NULL;
17     n = citire(&p);
18     for(i = 0; i < n; i++)
19         printf("p[%d] = %d\n",i,p[i]);
20
21     return 0;
22 }
```

```
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc alocareDinamica3.c
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
n=5
10
20
30
40
50
p[0] = 10
p[1] = 20
p[2] = 30
p[3] = 40
p[4] = 50
```

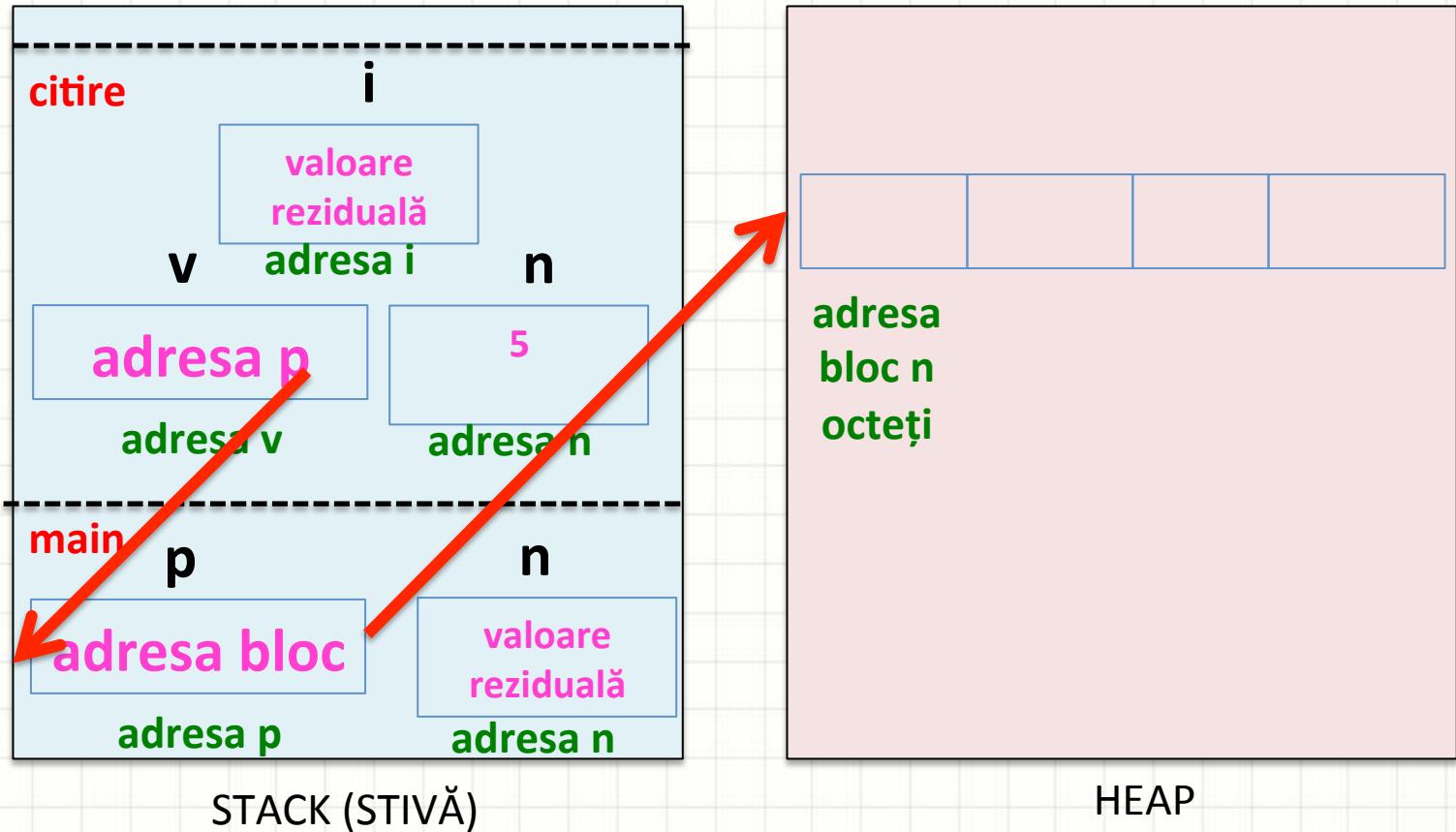
# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.



# Functia malloc

- exemplu: scriu o funcție pentru citirea unui tablou unidimensional. În interiorul funcției citesc numărul n de elemente, aloc dinamic tabloul și citesc elementele tabloului.



# Funcția calloc

- **prototipul funcției:**

***void \* calloc( int numar, int dimensiune);***

unde:

- **numar** = numărul de blocuri/elemente a se aloca
- **dimensiune** = numărul de octeți ceruți pentru fiecare bloc
- dacă există suficient spațiu liber în HEAP atunci un bloc de memorie continuu de dimensiunea specificată va fi marcat ca ocupat, iar funcția **calloc va returna un pointer ce conține adresa de început a acelui bloc**. Dacă nu există suficient spațiu liber funcția **calloc** întoarce NULL.
- diferența față de **malloc**: funcția **calloc** initializează toate blocurile cu 0 (vector de frecvențe) + nu face overflow (la malloc avem o înmulțire).

# Functia calloc

## □ exemplu:

```
alocareDinamica4.c  x

1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main()
5 {
6
7     int n,i,*p1 = NULL;
8     double *p2 = NULL;
9     char *p3 = NULL;
10
11    printf("n = "); scanf("%d",&n);
12
13    p1 = (int*) calloc(n,sizeof(int));
14    printf("\n Afisare adrese + valori vector de int alocat cu calloc \n");
15    for(i=0;i<n;i++)
16        printf("%p %d ", p1+i, p1[i]);
17
18    p2 = (double*) calloc(n,sizeof(double));
19    printf("\n Afisare adrese + valori vector de double alocat cu calloc \n");
20    for(i=0;i<n;i++)
21        printf("%p %f ", p2+i, p2[i]);
22
23    p3 = (char*) calloc(n,sizeof(char));
24    printf("\n Afisare adrese + valori vector de char alocat cu calloc \n");
25    for(i=0;i<n;i++)
26        printf("%p %d ", p3+i, p3[i]);
27    printf("\n");
28
29    return 0;
30 }
```

# Functia calloc

## □ exemplu:

```
alocareDinamica4.c  x

1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main()
5 {
6
7     int n,i,*p1 = NULL;
8     double *p2 = NULL;
9     char *p3 = NULL;
10
11    printf("n = "); scanf("%d",&n);
12
13    p1 = (int*) calloc(n,sizeof(int));
14    printf("\n Afisare adrese + valori vector de int alocat cu calloc \n");
15    for(i=0;i<n;i++)
16        printf("%p %d ", p1+i, p1[i]);
17
18    p2 = (double*) calloc(n,sizeof(double));
19    printf("\n Afisare adrese + valori vector de double alocat cu calloc \n");
20    for(i=0;i<n;i++)
21        printf("%p %f ", p2+i, p2[i]);
22
23    p3 = (char*) calloc(n,sizeof(char));
24    printf("\n Afisare adrese + valori vector de char alocat cu calloc \n");
25    for(i=0;i<n;i++)
26        printf("%p %c ", p3+i, p3[i]);
27    printf("\n");
28
29    return 0;
30 }
```

Bogdan-Alexes-MacBook-Pro:curs8 bogdan\$ gcc alocareDinamica4.c  
Bogdan-Alexes-MacBook-Pro:curs8 bogdan\$ ./a.out

|       |                                                                         |
|-------|-------------------------------------------------------------------------|
| n = 3 | Afisare adrese + valori vector de int alocat cu calloc                  |
|       | 0x7fa38a500000 0 0x7fa38a500004 0 0x7fa38a500008 0                      |
|       | Afisare adrese + valori vector de double alocat cu calloc               |
|       | 0x7fa38a500010 0.000000 0x7fa38a500018 0.000000 0x7fa38a500020 0.000000 |
|       | Afisare adrese + valori vector de char alocat cu calloc                 |
|       | 0x7fa38a5000a0 0 0x7fa38a5000a1 0 0x7fa38a5000a2 0                      |

# Funcția realloc

- ❑ **prototipul funcției:**

**void \* realloc( void \*p, int dimensiune);**

unde:

- ❑ **p** reprezinta un pointer (începutul unui bloc de memorie pe care vreau să îl redimensionez - fie să micșorez dimensiunea blocului, fie să o creasc – de obicei avem nevoie de mai multă memorie)
- ❑ **dimensiune** = numărul de octeți ceruți pentru alocare
- ❑ dacă există suficient spațiu liber în HEAP atunci un bloc de memorie continuu de dimensiunea specificată va fi marcat ca ocupat, iar funcția **realloc va returna un pointer ce conține adresa de început a acelui bloc. Tot conținutul blocului de memorie inițial se copiază.** Dacă nu există suficient spațiu liber **realloc** întoarce NULL.

# Functia realloc

## □ exemplu:

```
alocareDinamica5.c  x
00
01 1 #include<stdio.h>
02 2 #include<stdlib.h>
03
04 3 int main()
05 4 {
06
07 5     int *a, *aux;
08 6     a = (int *) malloc(100 * sizeof(int));
09 7     if(!a)
10 8     {
11 9         printf("Nu pot aloca memorie");
1210     return 1;
1311 }
14
1512 aux = (int *) realloc(a,200 * sizeof(int));
16
1713 if(!aux)
1814 {
1915     printf("Nu pot dimensiona blocul a");
2016     free(a);
2117     return 1;
2218 }
2319 printf("Redimensionare reusita \n");
2420 a = aux;
25
2621 free(a);
2722
2823     return 0;
29 }
```

```
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc alocareDinamica5.c
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Redimensionare reusita
```

# Functia free

- ❑ **prototipul functiei:**

**void free( void \*p);**

unde:

- ❑ **p** reprezinta un pointer (începutul unui bloc de memorie pe care vrem să-l eliberăm)
- ❑ functia **free** elibereaza zona de memoria alocata dinamic a cărei adresa de început este data de p. Zona de memorie dezalocata este marcată ca fiind disponibila pentru o nouă alocare.
- ❑ un bloc de memorie nu trebuie eliberat de mai multe ori.

# Alocare dinamică – aplicații

- ❑ principalul avantaj al folosirii alocării dinamice este gestionarea eficientă a resurselor memoriei. Memoria necesară este alocată în timpul execuției programului (când e nevoie) și nu la compilarea programului
- ❑ **exemplu:** se citește de la tastatură un sir de numere întregi până la întâlnirea lui 0. Să se afișeze numerele în ordinea inversă a citirii.

# Alocare dinamică – aplicații

- **exemplu:** se citește de la tastatură un sir de numere întregi până la întâlnirea lui 0. Să se afișeze numerele în ordinea inversă a citirii.

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

void afisare(int *p, int dim)
{
    printf("\nDupa %d reallocari: ",dim);
    for(int i = dim; i >= 0; i--)
        printf("%d\t", p[i]);
}

int main()
{
    int *p, *aux, i=0, valoareCitita;
    printf("Citim un nou numar: ");
    scanf("%d",&valoareCitita);
    p = (int*) malloc(sizeof(int));
    while(valoareCitita)
    {
        p[i] = valoareCitita;
        afisare(p,i); i++;
        p = realloc(p, (i+1)*sizeof(int));
        printf("\nCitim un nou numar: ");
        scanf("%d",&valoareCitita);
    }
    free(p);
    return 0;
}
```

Ce se întâmplă dacă nu pot să realloc memorie?  
p devine NULL (am pierdut tot conținutul de până atunci).

# Alocare dinamică – aplicații

- exemplu: se citește de la tastatură un sir de numere întregi până la întâlnirea lui 0. Să se afișeze numerele în ordinea inversă a citirii.

```
alocareDinamica7.c  x

1 #include<stdio.h>
2 #include<stdlib.h>
3
4 void afisare(int *p, int dim)
5 {
6     printf("\nDupa %d reallocari: ",dim);
7     for(int i = dim; i >= 0; i--)
8         printf("%d\t", p[i]);
9 }
10
11 int main()
12 {
13     int *p, *aux, i=0, valoareCitita;
14     printf("Citim un nou numar: ");
15     scanf("%d",&valoareCitita);
16     p = (int*) malloc(sizeof(int));
17     while(valoareCitita)
18     {
19         p[i] = valoareCitita;
20         afisare(p,i); i++;
21         int* aux = realloc(p, (i+1)*sizeof(int));
22         if(!aux)
23         {
24             printf("Eroare la reallocare");
25             free(p);
26             return 1;
27         }
28         p = aux;
29         printf("\nCitim un nou numar: ");
30         scanf("%d",&valoareCitita);
31     }
32     free(p);
33     return 0;
34 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc alocareDinamica6.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Citim un nou numar: 10
Dupa 0 reallocari: 10
Citim un nou numar: 20
Dupa 1 reallocari: 20      10
Citim un nou numar: 30
Dupa 2 reallocari: 30      20      10
Citim un nou numar: 40
Dupa 3 reallocari: 40      30      20      10
Citim un nou numar: 50
Dupa 4 reallocari: 50      40      30      20      10
Citim un nou numar: 0
```

# Alocare dinamică – aplicații

- **exemplu:** se citește de la tastatură un sir de numere întregi până la întâlnirea lui 0. Să se afișeze numerele în ordinea inversă a citirii.

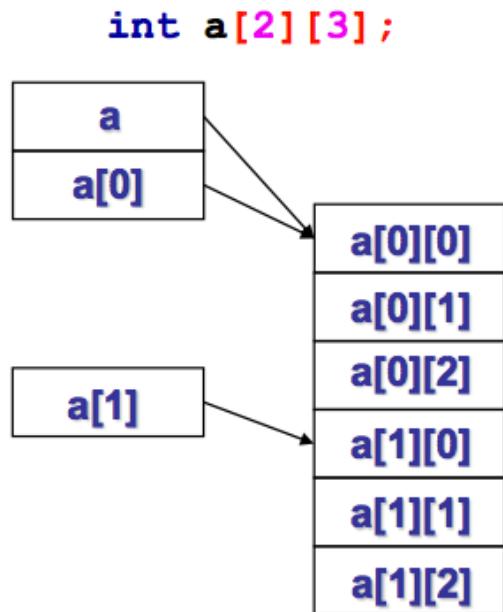
```
alocareDinamica7.c
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 void afisare(int *p, int dim)
5 {
6     printf("\nDupa %d realocari: ",dim);
7     for(int i = dim; i >= 0; i--)
8         printf("%d\t", p[i]);
9 }
10
11 int main()
12 {
13     int *p, *aux, i=0, valoareCitita;
14     printf("Citim un nou numar: ");
15     scanf("%d",&valoareCitita);
16     p = (int*) malloc(sizeof(int));
17     while(valoareCitita)
18     {
19         p[i] = valoareCitita;
20         afisare(p,i); i++;
21         int* aux = realloc(p, (i+1)*sizeof(int));
22         if(!aux)
23         {
24             printf("Eroare la reallocare");
25             free(p);
26             return 1;
27         }
28         p = aux;
29         printf("\nCitim un nou numar: ");
30         scanf("%d",&valoareCitita);
31     }
32     free(p);
33     return 0;
34 }
```

Dacă vreau să citesc 10000 de elemente (eventual dintr-un fisier) programul e mult prea lent. Soluția mai eficientă este să folosesc un buffer de 1000 de elemente.

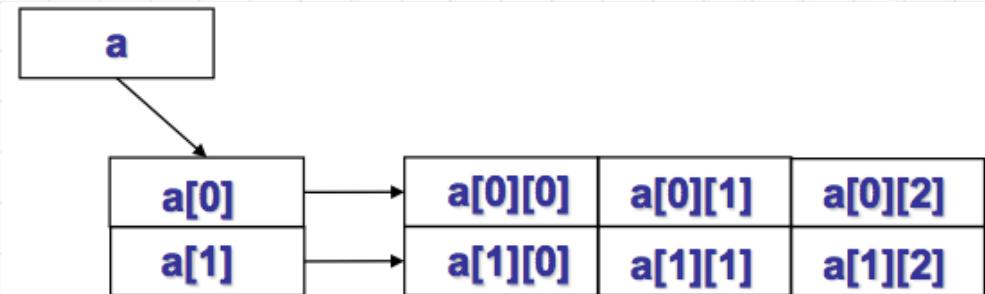
# Alocare dinamică – aplicații

- exemplu: alocarea dinamică a unui tablou bi-dimensional

## Alocarea statică (pe STIVĂ)



## Alocarea dinamică (pe HEAP)



`a` e pointer dublu: `a` pointeaza catre un tablou de pointeri, fiecare pointer pointeaza catre o linie

# Pointeri la pointeri (pointeri dubli)

## □ sintaxa

**tip      `**nume_variabilă;`**

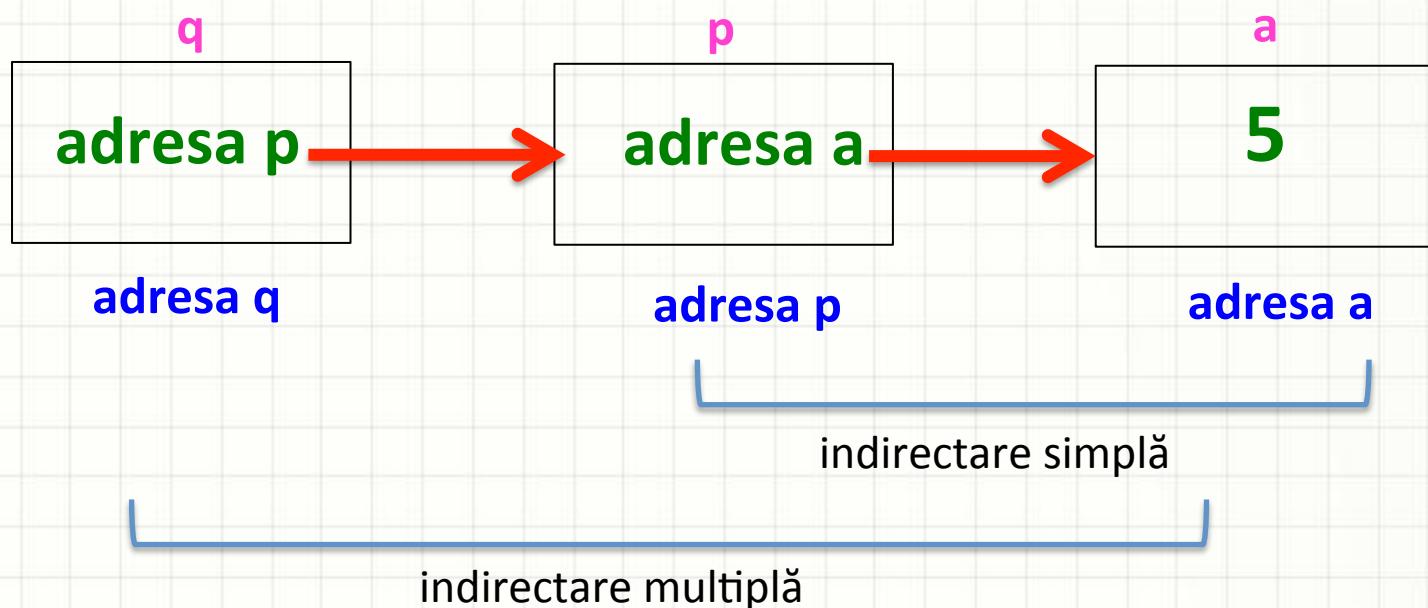
**tip** = tipul de bază al variabilei de tip pointer dublu nume\_variabilă;

**\*** = operator de indirectare;

**nume\_variabila** = variabila de tip pointer dublu care poate lua ca valori adrese de memorie ale unor variabile de tip pointer.

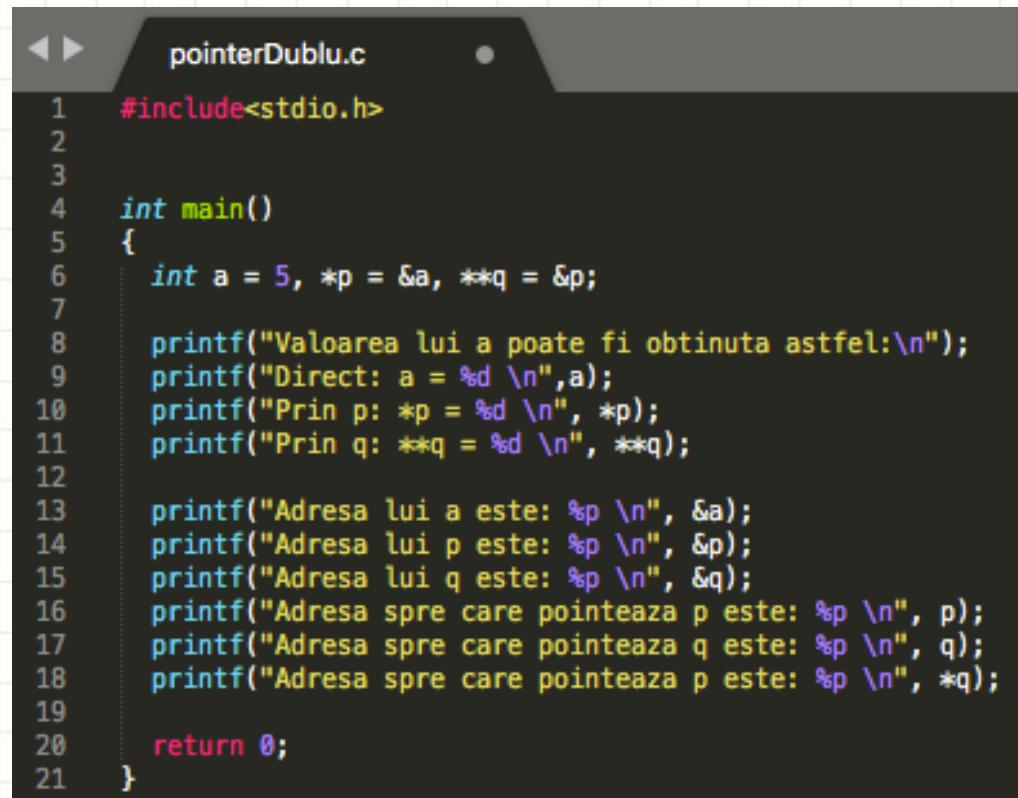
## □ exemplu:

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



# Pointeri la pointeri

## □ exemplu:



The image shows a code editor window with a dark theme. The title bar of the window says "pointerDublu.c". The code itself is written in C and demonstrates various levels of pointer usage. It includes declarations for variables, assignments, and multiple printf statements to print out memory addresses and values.

```
#include<stdio.h>

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

    printf("Valoarea lui a poate fi obtinuta astfel:\n");
    printf("Direct: a = %d \n", a);
    printf("Prin p: *p = %d \n", *p);
    printf("Prin q: **q = %d \n", **q);

    printf("Adresa lui a este: %p \n", &a);
    printf("Adresa lui p este: %p \n", &p);
    printf("Adresa lui q este: %p \n", &q);
    printf("Adresa spre care pointeaza p este: %p \n", p);
    printf("Adresa spre care pointeaza q este: %p \n", q);
    printf("Adresa spre care pointeaza p este: %p \n", *q);

    return 0;
}
```

# Pointeri la pointeri

## exemplu:

```
pointerDublu.c
1 #include<stdio.h>
2
3
4 int main()
5 {
6     int a = 5, *p = &a, **q = &p;
7
8     printf("Valoarea lui a poate fi obtinuta direct:\n");
9     printf("Direct: a = %d \n", a);
10    printf("Prin p: *p = %d \n", *p);
11    printf("Prin q: **q = %d \n", **q);
12
13    printf("Adresa lui a este: %p \n", &a);
14    printf("Adresa lui p este: %p \n", &p);
15    printf("Adresa lui q este: %p \n", &q);
16    printf("Adresa spre care pointeaza p este: %p \n", p);
17    printf("Adresa spre care pointeaza q este: %p \n", q);
18    printf("Adresa spre care pointeaza p este: %p \n", *p);
19
20    return 0;
21 }
```

```
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc pointerDublu.c
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Valoarea lui a poate fi obtinuta astfel:
Direct: a = 5
Prin p: *p = 5
Prin q: **q = 5
Adresa lui a este: 0x7fff5ef14b48
Adresa lui p este: 0x7fff5ef14b40
Adresa lui q este: 0x7fff5ef14b38
Adresa spre care pointeaza p este: 0x7fff5ef14b48
Adresa spre care pointeaza q este: 0x7fff5ef14b40
Adresa spre care pointeaza p este: 0x7fff5ef14b48
```

q

0x7fff5ef14b40

p

0x7fff5ef14b48

a

5

0x7fff5ef14b38

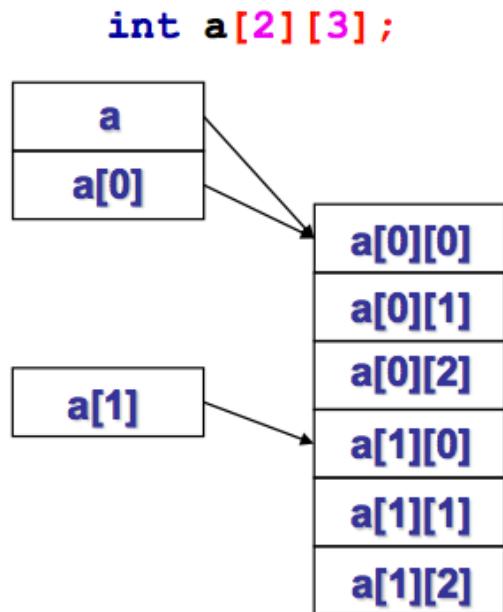
0x7fff5ef14b40

0x7fff5ef14b48

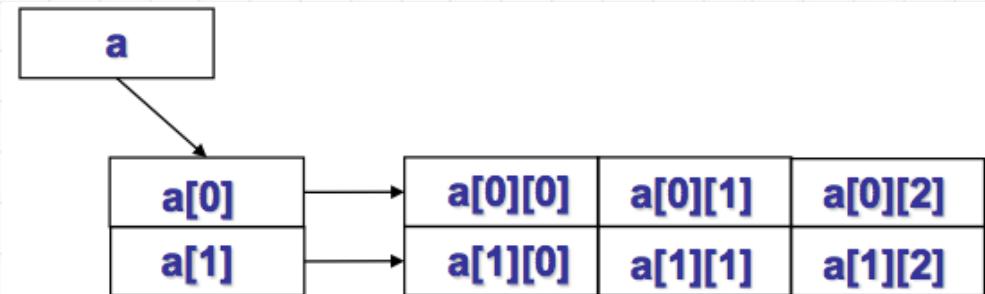
# Alocare dinamică – aplicații

- exemplu: alocarea dinamică a unui tablou bi-dimensional

## Alocarea statică (pe STIVĂ)



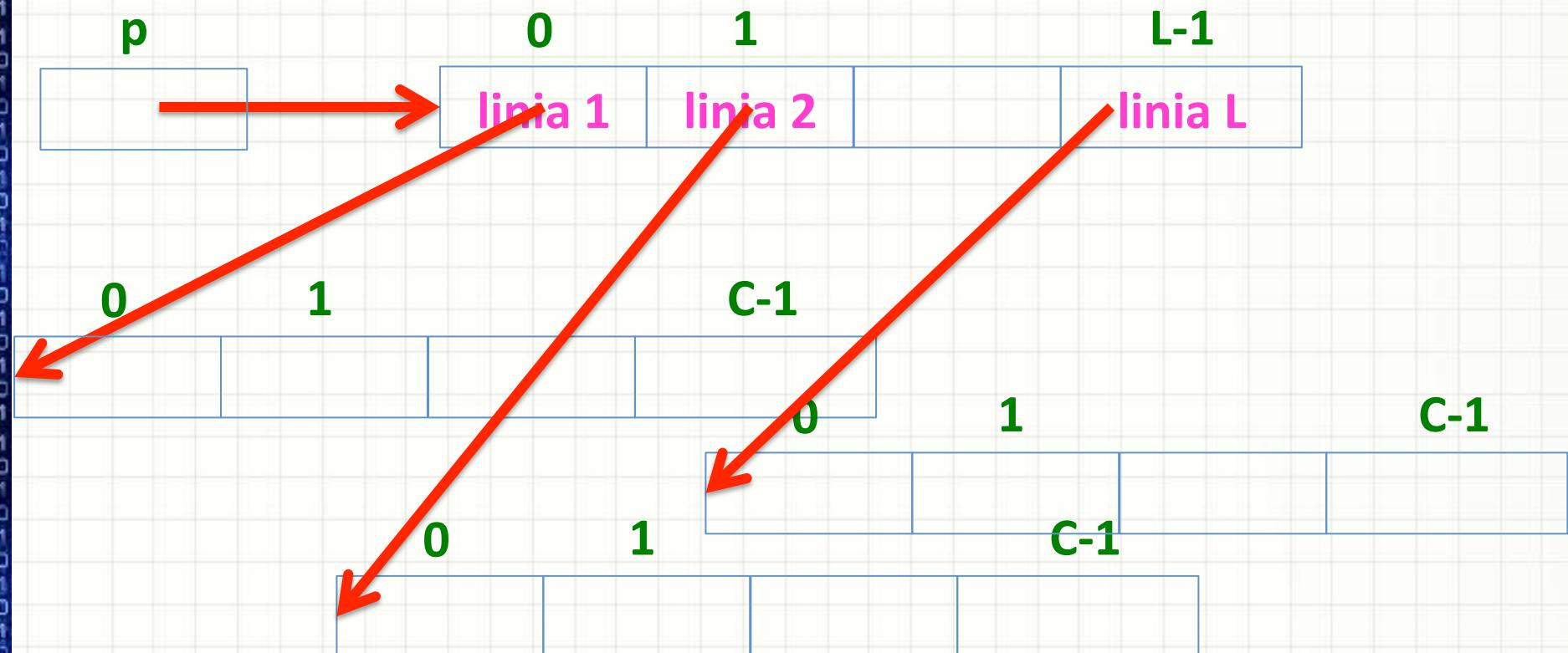
## Alocarea dinamică (pe HEAP)



`a` e pointer dublu: `a` pointează către un tablou de pointeri, fiecare pointer pointează către o linie

# Alocare dinamică – aplicații

- exemplu: alocarea dinamică a unui tablou bi-dimensional



# Alocare dinamică – aplicații

```
tablou_bidimensional.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int L, C, i, j;
6     int **p; // Adresa matrice
7
8     printf("Nr de linii L = "); scanf("%d", &L);
9     printf("Nr de coloane C = "); scanf("%d", &C);
10
11    p = (int**) malloc(L * sizeof(int*));
12    printf("Sizeof(int*) = %lu \n", sizeof(int*));
13    printf("Pointerul p contine adresa %p \n", p);
14
15    for (i = 0; i < L; i++)
16    {
17        p[i] = calloc(C, sizeof(int));
18        printf("Linia %d incepe la %p \n", i, p[i]);
19    }
20
21    for (i = 0; i < L; i++) {
22        for (j = 0; j < C; j++) {
23            p[i][j] = L * i + j + 1;
24            printf("Adresa lui p[%d][%d] este = %p \n", i, j, &p[i][j]);
25        }
26    }
27
28    for (i = 0; i < L; i++) {
29        for (j = 0; j < C; j++) {
30            printf("%d ", p[i][j]);
31        }
32        printf("\n");
33    }
34    return 0;
35 }
```

# Alocare dinamică – aplicații

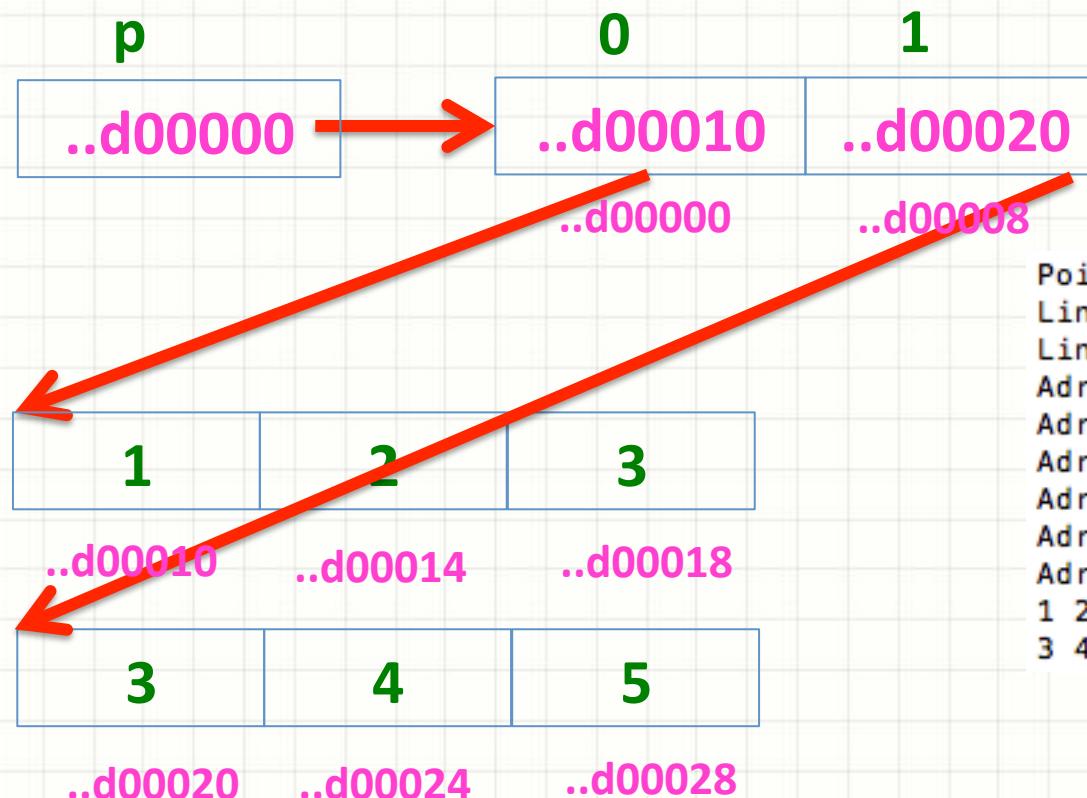
```
tablou_bidimensional.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int L, C, i, j;
6     int **p; // Adresa matrice
7
8     printf("Nr de linii L = "); scanf("%d", &L);
9     printf("Nr de coloane C = "); scanf("%d", &C);
10
11    p = (int**) malloc(L * sizeof(int*));
12    printf("Sizeof(int*) = %lu\n", sizeof(int*));
13    printf("Pointerul p contine %p\n", p);
14
15    for (i = 0; i < L; i++)
16    {
17        p[i] = calloc(C, sizeof(int));
18        printf("Linia %d incepe la %p\n", i, p[i]);
19    }
20
21    for (i = 0; i < L; i++) {
22        for (j = 0; j < C; j++) {
23            p[i][j] = L * i + j + 1;
24            printf("Adresa lui p[%d][%d] este %p\n", i, j, p[i][j]);
25        }
26    }
27
28    for (i = 0; i < L; i++) {
29        for (j = 0; j < C; j++) {
30            printf("%d ", p[i][j]);
31        }
32        printf("\n");
33    }
34
35    return 0;
}
```

```
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc tablou_bidimensional.c
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Nr de linii L = 2
Nr de coloane C = 3
Sizeof(int*) = 8
Pointerul p contine adresa 0x7fe977d00000
Linia 0 incepe la 0x7fe977d00010
Linia 1 incepe la 0x7fe977d00020
Adresa lui p[0][0] este = 0x7fe977d00010
Adresa lui p[0][1] este = 0x7fe977d00014
Adresa lui p[0][2] este = 0x7fe977d00018
Adresa lui p[1][0] este = 0x7fe977d00020
Adresa lui p[1][1] este = 0x7fe977d00024
Adresa lui p[1][2] este = 0x7fe977d00028
1 2 3
3 4 5
```

# Alocare dinamică – aplicații

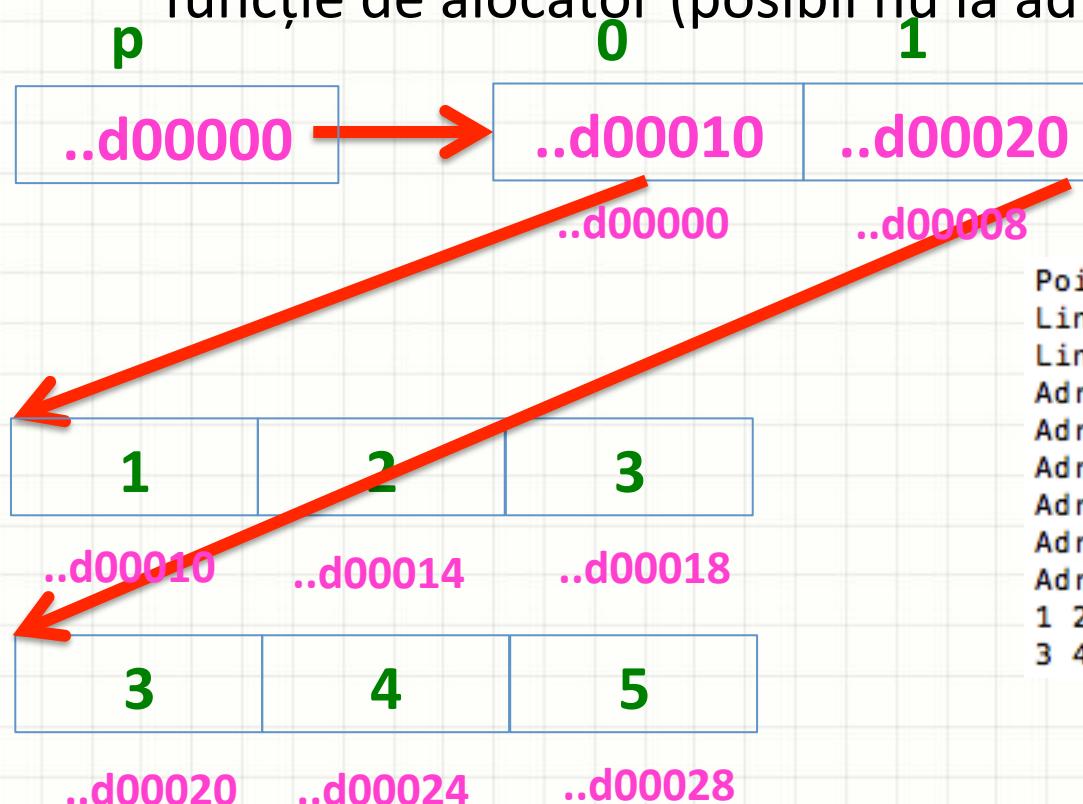
- exemplu: alocarea dinamică a unui tablou bi-dimensional



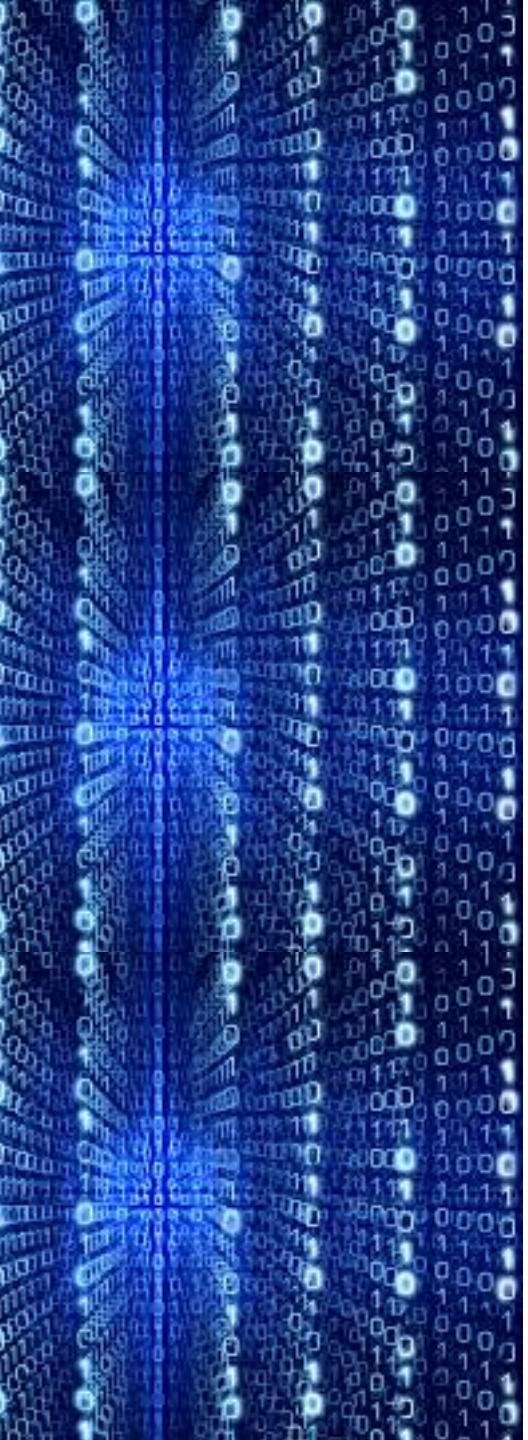
```
Pointerul p contine adresa 0x7fe977d00000
Linia 0 incepe la 0x7fe977d00010
Linia 1 incepe la 0x7fe977d00020
Adresa lui p[0][0] este = 0x7fe977d00010
Adresa lui p[0][1] este = 0x7fe977d00014
Adresa lui p[0][2] este = 0x7fe977d00018
Adresa lui p[1][0] este = 0x7fe977d00020
Adresa lui p[1][1] este = 0x7fe977d00024
Adresa lui p[1][2] este = 0x7fe977d00028
1 2 3
3 4 5
```

# Alocare dinamică – aplicații

- ❑ exemplu: alocarea dinamică a unui tablou bi-dimensional
  - ❑ alocare necompactă, toate liniile sunt puse în memorie în funcție de alocator (posibil nu la adrese consecutive)



Pointerul p conține adresa 0x7fe977d00000  
Linia 0 începe la 0x7fe977d00010  
Linia 1 începe la 0x7fe977d00020  
Adresa lui p[0][0] este = 0x7fe977d00010  
Adresa lui p[0][1] este = 0x7fe977d00014  
Adresa lui p[0][2] este = 0x7fe977d00018  
Adresa lui p[1][0] este = 0x7fe977d00020  
Adresa lui p[1][1] este = 0x7fe977d00024  
Adresa lui p[1][2] este = 0x7fe977d00028  
1 2 3  
3 4 5



# PROGRAMARE PROCEDURALĂ

Bogdan Alexe

[bogdan.alexe@fmi.unibuc.ro](mailto:bogdan.alexe@fmi.unibuc.ro)

Secția Informatică, anul I,

2018-2019

Cursul 9

# 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, siruri 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.
  - Fișiere binare**
    - Funcții specifice de manipulare.
  - Funcții (1)**
    - Declarare și definire. Apel. Metode de transmitere a parametrilor. Pointeri la funcții.
  - Tablouri și pointeri**
    - Aritmetică pointerilor
    - Legătura dintre tablouri și pointeri
    - Alocarea dinamică a memoriei
    - Clase de memorare
  - Siruri de caractere**
    - 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**
- 

# Cuprinsul cursului de azi

1. Alocarea dinamică a memoriei
2. Clase de memorare
3. Siruri de caractere – funcții specifice de manipulare

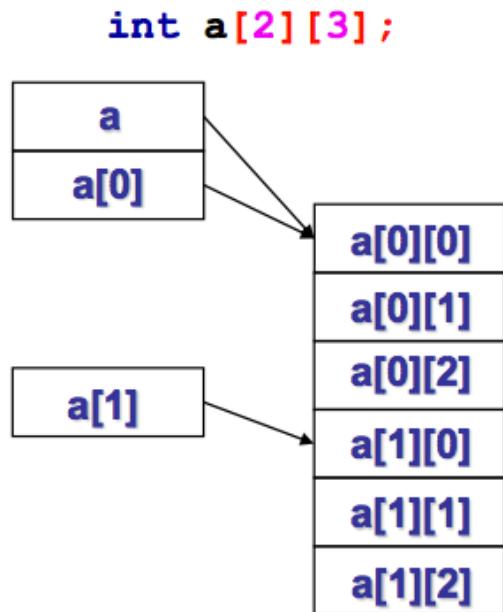
# Alocarea dinamică a memoriei

- *heap*-ul este o zonă predefinită de memorie (de dimensiuni foarte mari) care poate fi accesată de program pentru a stoca date și variabile;
- datele și variabilele pot fi alocate pe *heap* prin apeluri speciale de funcții din biblioteca *stdlib.h*: **malloc**, **calloc**, **realloc**
- zonele de memorie pot să fie dezalocate la cerere prin apelul funcției **free**
- este recomandat ca memoria să fie eliberată în momentul în care datele/variabilele respective nu mai sunt de interes!

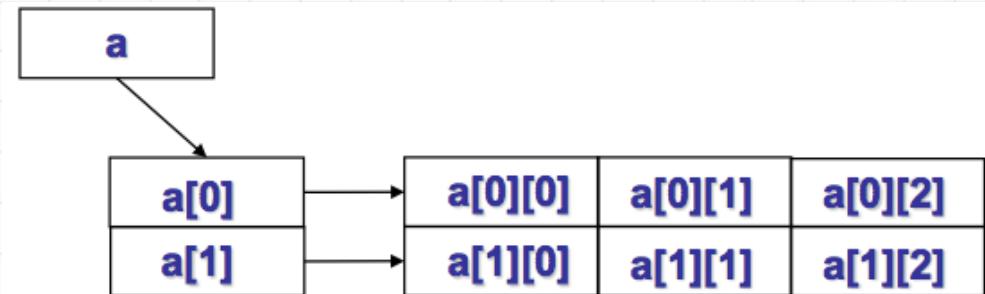
# Alocare dinamică – aplicații

- exemplu: alocarea dinamică a unui tablou bi-dimensional

## Alocarea statică (pe STIVĂ)



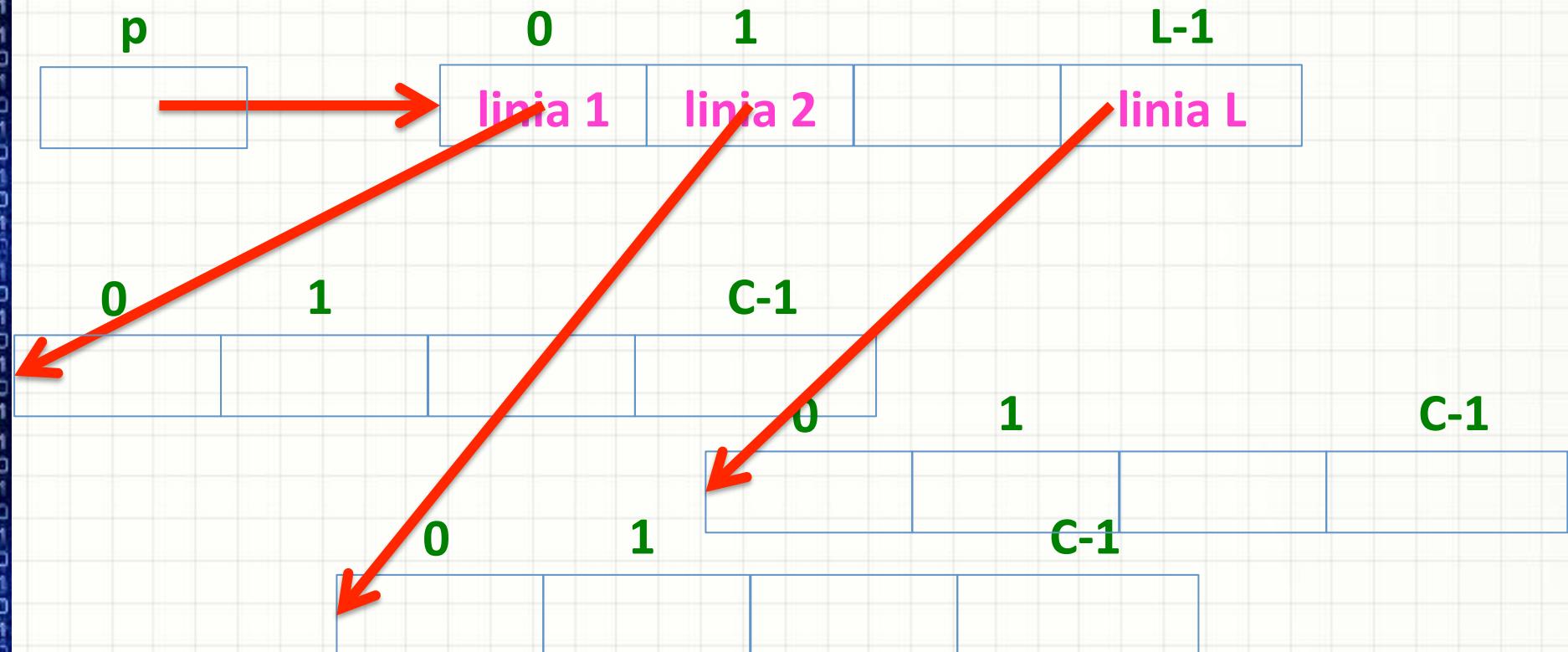
## Alocarea dinamică (pe HEAP)



`a` e pointer dublu: `a` pointeaza catre un tablou de pointeri, fiecare pointer pointeaza catre o linie

# Alocare dinamică – aplicații

- exemplu: alocarea dinamică a unui tablou bi-dimensional



# Alocare dinamică – aplicații

```
tablou_bidimensional.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int L, C, i, j;
6     int **p; // Adresa matrice
7
8     printf("Nr de linii L = "); scanf("%d", &L);
9     printf("Nr de coloane C = "); scanf("%d", &C);
10
11    p = (int**) malloc(L * sizeof(int*));
12    printf("Sizeof(int*) = %lu \n", sizeof(int*));
13    printf("Pointerul p contine adresa %p \n", p);
14
15    for (i = 0; i < L; i++)
16    {
17        p[i] = calloc(C, sizeof(int));
18        printf("Linia %d incepe la %p \n", i, p[i]);
19    }
20
21    for (i = 0; i < L; i++) {
22        for (j = 0; j < C; j++) {
23            p[i][j] = L * i + j + 1;
24            printf("Adresa lui p[%d][%d] este = %p \n", i, j, &p[i][j]);
25        }
26    }
27
28    for (i = 0; i < L; i++) {
29        for (j = 0; j < C; j++) {
30            printf("%d ", p[i][j]);
31        }
32        printf("\n");
33    }
34    return 0;
35 }
```

# Alocare dinamică – aplicații

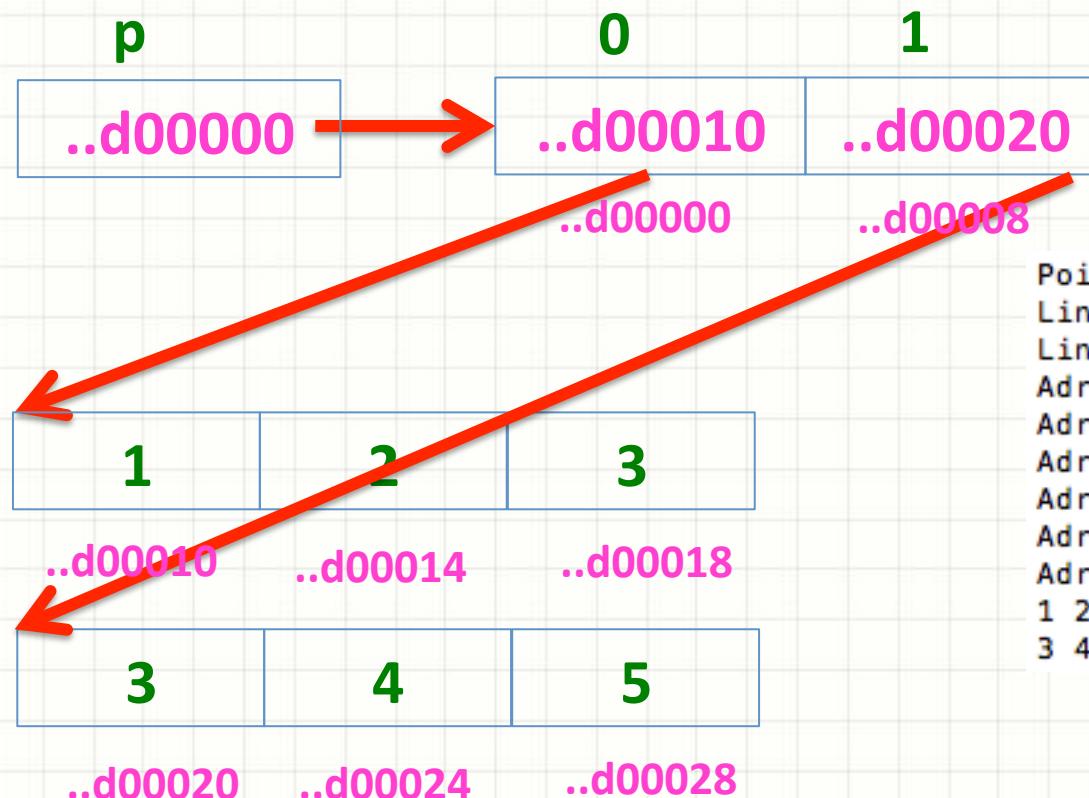
```
tablou_bidimensional.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int L, C, i, j;
6     int **p; // Adresa matrice
7
8     printf("Nr de linii L = "); scanf("%d", &L);
9     printf("Nr de coloane C = "); scanf("%d", &C);
10
11    p = (int**) malloc(L * sizeof(int*));
12    printf("Sizeof(int*) = %lu\n", sizeof(int*));
13    printf("Pointerul p contine %p\n", p);
14
15    for (i = 0; i < L; i++)
16    {
17        p[i] = calloc(C, sizeof(int));
18        printf("Linia %d incepe la %p\n", i, p[i]);
19    }
20
21    for (i = 0; i < L; i++) {
22        for (j = 0; j < C; j++) {
23            p[i][j] = L * i + j + 1;
24            printf("Adresa lui p[%d][%d] este %p\n", i, j, p[i][j]);
25        }
26    }
27
28    for (i = 0; i < L; i++) {
29        for (j = 0; j < C; j++) {
30            printf("%d ", p[i][j]);
31        }
32        printf("\n");
33    }
34
35    return 0;
}
```

```
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc tablou_bidimensional.c
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Nr de linii L = 2
Nr de coloane C = 3
Sizeof(int*) = 8
Pointerul p contine adresa 0x7fe977d00000
Linia 0 incepe la 0x7fe977d00010
Linia 1 incepe la 0x7fe977d00020
Adresa lui p[0][0] este = 0x7fe977d00010
Adresa lui p[0][1] este = 0x7fe977d00014
Adresa lui p[0][2] este = 0x7fe977d00018
Adresa lui p[1][0] este = 0x7fe977d00020
Adresa lui p[1][1] este = 0x7fe977d00024
Adresa lui p[1][2] este = 0x7fe977d00028
1 2 3
3 4 5
```

# Alocare dinamică – aplicații

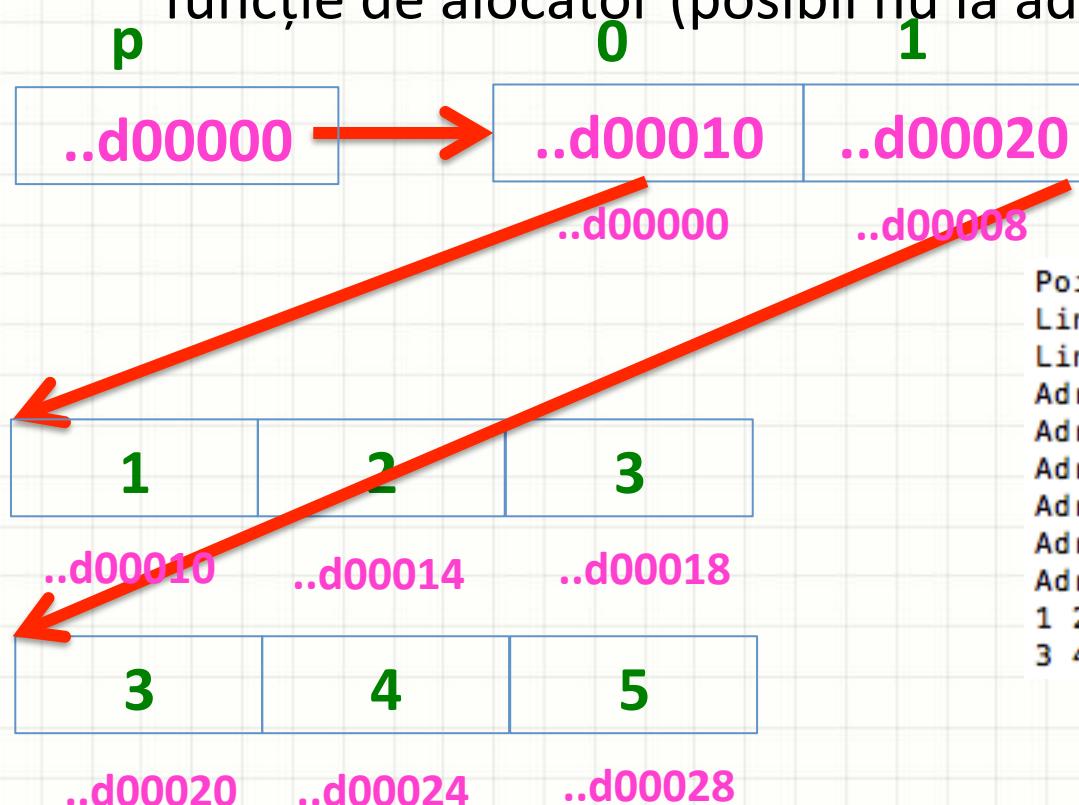
- exemplu: alocarea dinamică a unui tablou bi-dimensional



```
Pointerul p contine adresa 0x7fe977d00000
Linia 0 incepe la 0x7fe977d00010
Linia 1 incepe la 0x7fe977d00020
Adresa lui p[0][0] este = 0x7fe977d00010
Adresa lui p[0][1] este = 0x7fe977d00014
Adresa lui p[0][2] este = 0x7fe977d00018
Adresa lui p[1][0] este = 0x7fe977d00020
Adresa lui p[1][1] este = 0x7fe977d00024
Adresa lui p[1][2] este = 0x7fe977d00028
1 2 3
3 4 5
```

# Alocare dinamică – aplicații

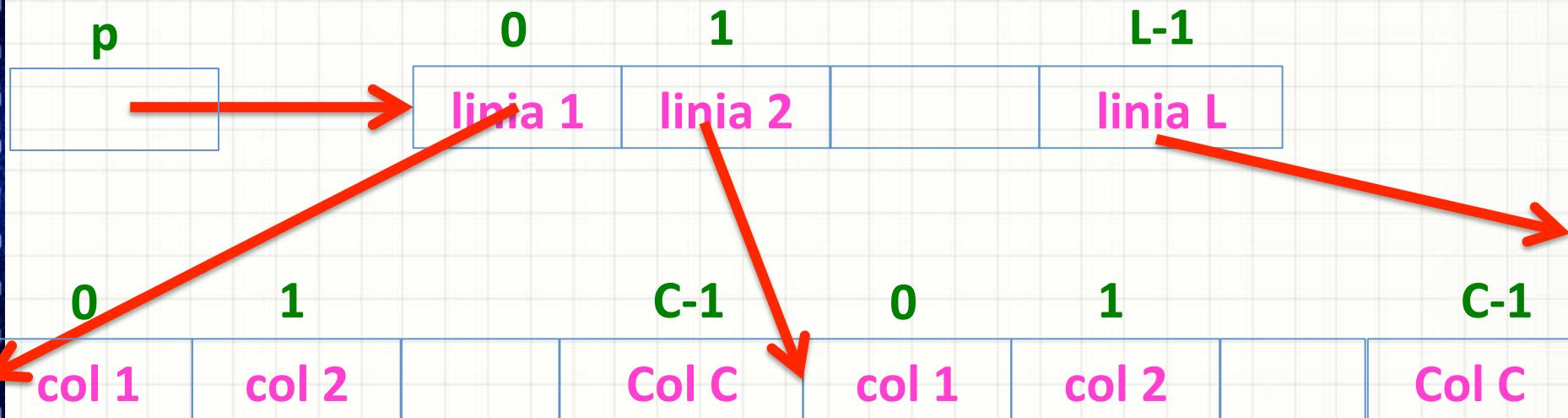
- ❑ exemplu: alocarea dinamică a unui tablou bi-dimensional
  - ❑ alocare necompactă, toate liniile sunt puse în memorie în funcție de alocator (posibil nu la adrese consecutive)



```
Pointerul p contine adresa 0x7fe977d00000
Linia 0 incepe la 0x7fe977d00010
Linia 1 incepe la 0x7fe977d00020
Adresa lui p[0][0] este = 0x7fe977d00010
Adresa lui p[0][1] este = 0x7fe977d00014
Adresa lui p[0][2] este = 0x7fe977d00018
Adresa lui p[1][0] este = 0x7fe977d00020
Adresa lui p[1][1] este = 0x7fe977d00024
Adresa lui p[1][2] este = 0x7fe977d00028
1 2 3
3 4 5
```

# Alocare dinamică – aplicații

- ❑ exemplu: alocarea dinamică a unui tablou bi-dimensional
  - ❑ soluție cu alocare compactă, toate elementelor liniilor puse unele după altele, ca în STIVĂ



# Alocare dinamică – aplicații

tablou2d\_compact.c x

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int L, C, i, j;
6     int **p; // Adresa matrice
7
8     printf("Nr de linii L = "); scanf("%d", &L);
9     printf("Nr de coloane C = "); scanf("%d", &C);
10
11    p = (int**) malloc(L * sizeof(int*));
12    printf("Sizeof(int*) = %lu \n", sizeof(int*));
13    printf("Pointerul p contine adresa %p \n", p);
14
15    p[0] = (int*) calloc(C*L, sizeof(int));
16
17    for (i = 0; i < L; i++)
18    {
19        p[i] = p[0] + i*C;
20        printf("Linia %d incepe la %p \n", i+1, p[i]);
21    }
22
23    for (i = 0; i < L; i++) {
24        for (j = 0; j < C; j++) {
25            p[i][j] = i + j + 1;
26            printf("Adresa lui p[%d][%d] este = %p \n", i, j, &p[i][j]);
27        }
28    }
29
30    for (i = 0; i < L; i++) {
31        for (j = 0; j < C; j++) {
32            printf("%d ", p[i][j]);
33        }
34        printf("\n");
35    }
36
37    return 0;
38 }
```

tablou bi-dimensional  
de elementelor liniilor puse

# Alocare dinamică – aplicații

tablou2d\_compact.c x

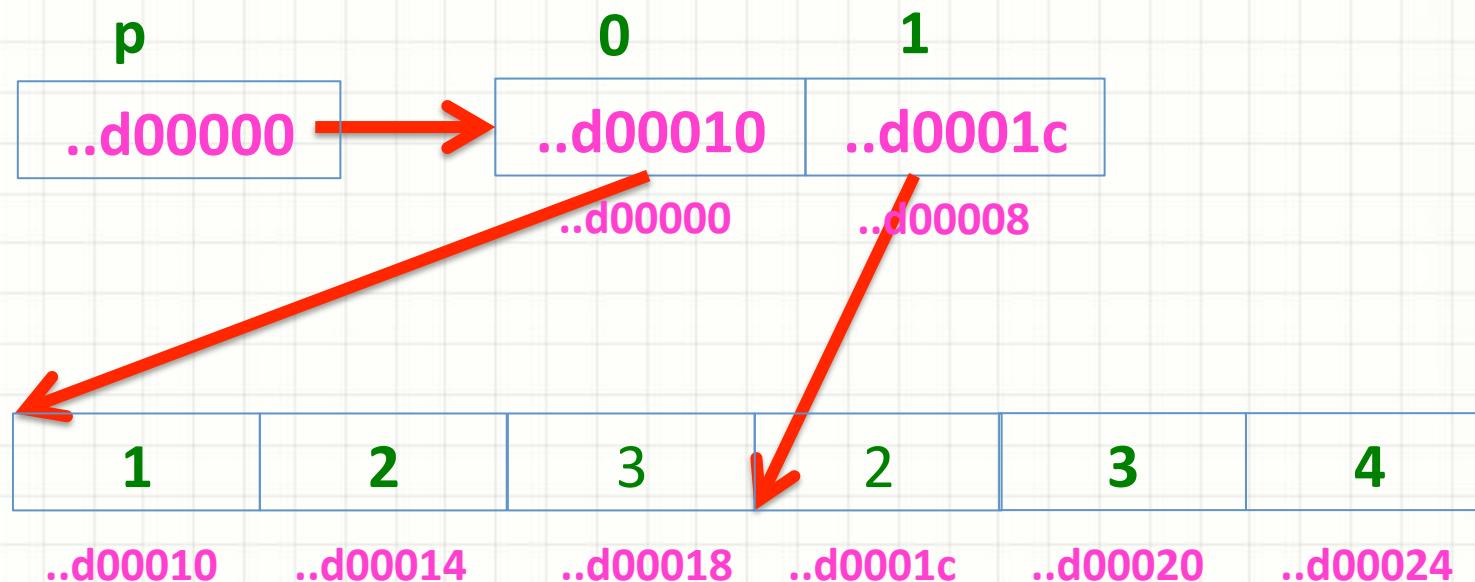
```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int L, C, i, j;
6     int **p; // Adresa matrice
7
8     printf("Nr de linii L = "); scanf("%d", &L);
9     printf("Nr de coloane C = "); scanf("%d", &C);
10
11    p = (int**) malloc(L * sizeof(int*));
12    printf("Sizeof(int*) = %lu \n", sizeof(int*));
13    printf("Pointerul p contine adresa %p \n", p);
14
15    p[0] = (int*) calloc(C*L, sizeof(int));
16
17    for (i = 0; i < L; i++)
18    {
19        p[i] = p[0] + i*C;
20        printf("Linia %d incepe la %p \n", i, p[i]);
21    }
22
23    for (i = 0; i < L; i++) {
24        for (j = 0; j < C; j++) {
25            p[i][j] = i + j + 1;
26            printf("Adresa lui p[%d][%d] este %p \n", i, j, p[i][j]);
27        }
28    }
29
30    for (i = 0; i < L; i++) {
31        for (j = 0; j < C; j++) {
32            printf("%d ", p[i][j]);
33        }
34        printf("\n");
35    }
36
37    return 0;
38 }
```

tablou bi-dimensional  
de elementelor liniilor puse

```
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc tablou2d_compact.c
Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Nr de linii L = 2
Nr de coloane C = 3
Sizeof(int*) = 8
Pointerul p contine adresa 0x7f8b99d00000
Linia 1 incepe la 0x7f8b99d00010
Linia 2 incepe la 0x7f8b99d0001c
Adresa lui p[0][0] este = 0x7f8b99d00010
Adresa lui p[0][1] este = 0x7f8b99d00014
Adresa lui p[0][2] este = 0x7f8b99d00018
Adresa lui p[1][0] este = 0x7f8b99d0001c
Adresa lui p[1][1] este = 0x7f8b99d00020
Adresa lui p[1][2] este = 0x7f8b99d00024
1 2 3
2 3 4
```

# Alocare dinamică – aplicații

- ❑ exemplu: alocarea dinamică a unui tablou bi-dimensional
  - ❑ soluție cu alocare compactă, toate elementelor liniilor puse unele după altele, ca în STIVĂ



# Alocare dinamică – aplicații

- **problemă la seminar/laborator:** alocarea dinamică a matricelor inferior/superior triunghiulare

Scriți un program care citește de la tastatură două matrice: una inferior triunghiulară (toate elementele de deasupra diagonalei principale sunt nule) și una superior triunghiulară (toate elementele de sub diagonala principală sunt nule). Ele vor fi stocate în memorie folosind cât mai puțin spațiu (fără a memora zerourile de deasupra/dedesubtul diagonalei principale). Calculați produsul celor două matrice.

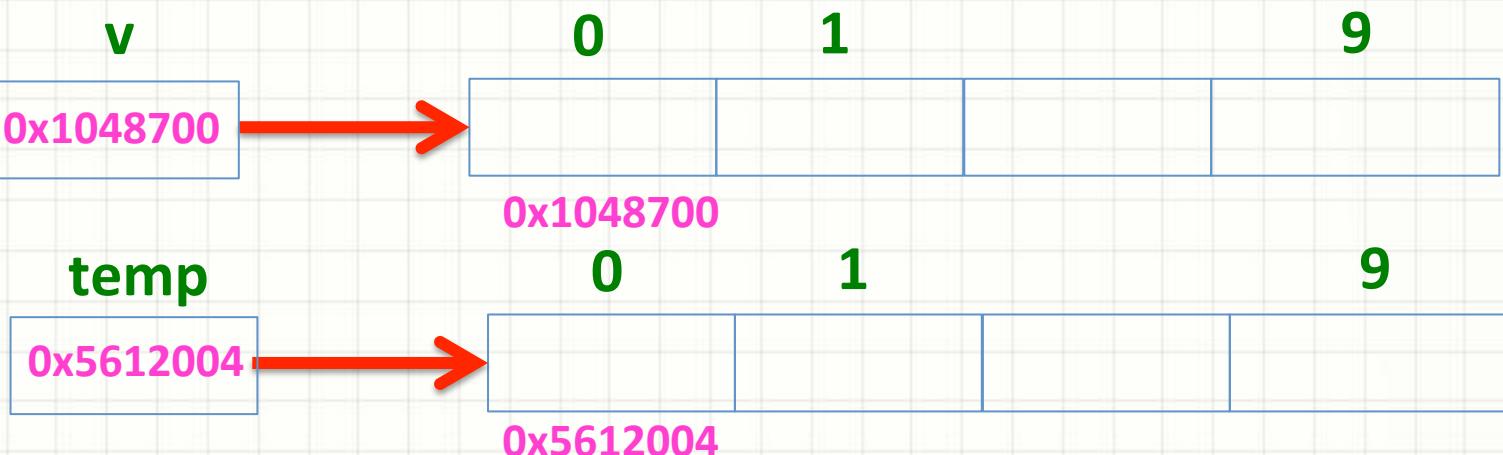
- **problemă la seminar/laborator:** interschimbarea de linii într-o matrice: alocare dinamică cu pointer dublu vs. alocare statică

# Alocare dinamică – avantaje + dezavantaje

- **avantaje:**
  - **durata de viață:** putem controla când are loc alocarea și dezalocarea memoriei
  - **memoria:** dimensiunea memoriei alocată poate fi controlată în timpul execuției programului. Spre exemplu un tablou poate fi alocat astfel încât are să aibă dimensiunea identică cu cea a unui tablou specificat în timpul execuției programului
- **dezavantaje:**
  - **mai mult de codat:** alocarea memoriei trebuie făcută explicit în cod
  - **posibile bug-uri:** lucrul cu pointerii (crash-uri de memorie)

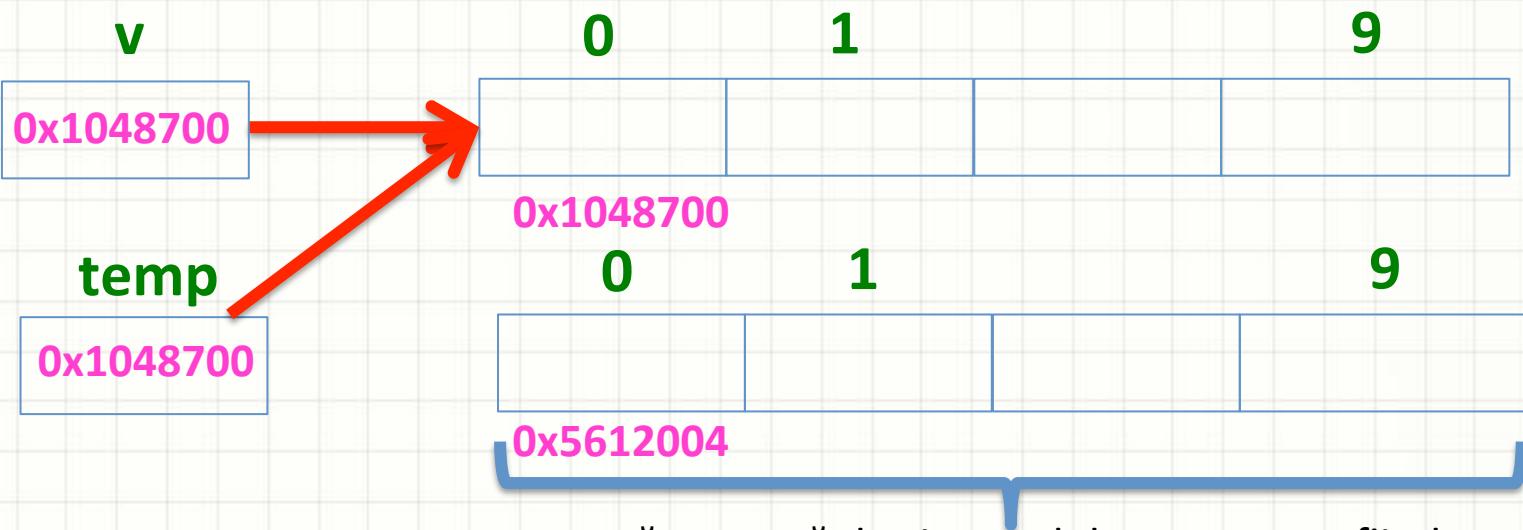
# Alocare dinamică – greșeli

```
int *v, *temp;  
v = (int*) malloc(10*sizeof(int));  
temp = (int*) malloc(10*sizeof(int));  
temp = v; //fac o copie a lui v in temp
```



# Alocare dinamică – greșeli

```
int *v, *temp;  
v = (int*) malloc(10*sizeof(int));  
temp = (int*) malloc(10*sizeof(int));  
temp = v; //fac o copie a lui v in temp
```



(zonă orfană de memorie)

# Alocare dinamică – greșeli

```
void f(...){  
int *p = (int*) malloc(10*sizeof(int));  
...  
}
```



**p este variabilă locală funcției f și va fi distrusă la ieșirea din funcție. Totuși memoria rămâne alocată și inutilizabilă (zonă orfană de memorie).**

```
void f(...){  
int *p = (int*) malloc(10*sizeof(int));  
free(p); //eliberare memorie  
}
```

# Alocare dinamică – greșeli

```
char nume[20] = "Paul";
```

...

```
char *t;  
strcpy(t,nume);
```

**t este un pointer fără zonă de memorie alocată. Va rezulta un crash de memorie.**

```
char nume[20] = "Paul";
```

...

```
char *t = malloc(strlen(nume) + 1);  
strcpy(t,nume);
```

# Transmiterea tablourilor 1D ca argumente funcțiilor

## Alocarea statică (pe STIVĂ)

```
int v[3];
```

## Alocarea dinamică (pe HEAP)

```
int *v;  
v = (int *) malloc(3 * sizeof(int));
```

- În ambele cazuri (alocare statică și alocare dinamică) se transmite adresa primului element + lungimea tabloului (nu am cum să o iau de altundeva)

```
int numeFunctie(int v[], int n)
```

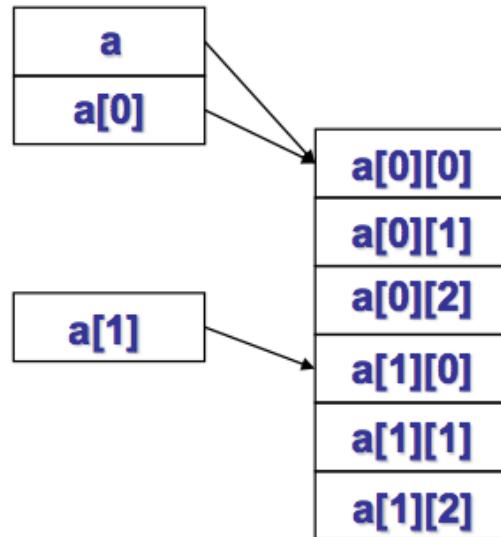
```
int numeFunctie(int v[10], int n)
```

```
int numeFunctie(int* v, int n)
```

# Transmiterea tablourilor 2D ca argumente funcțiilor

## Alocarea statică (pe STIVĂ)

```
int a[2][3];
```



- pentru tablouri 2D în alocare statică:
  - trebuie să transmit neapărat a doua dimensiune (compilatorul trebuie să știe câte elemente are o "linie")

# Transmiterea tablourilor 2D ca argumente funcțiilor

## Alocarea statică (pe STIVĂ)

```
exempluTransmitere_static.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void afiseazaMatrice(int x[][4], int nrLinii, int nrColoane)
5 {
6     int i,j;
7     for (i = 0; i < nrLinii; i++) {
8         for (j = 0;j < nrColoane; j++) {
9             printf("%d ", x[i][j]);
10        }
11        printf("\n");
12    }
13 }
14
15
16 int main() {
17     int i ,j, L = 3, C = 4;
18     int p[3][4];
19
20
21     for (i = 0; i < 3; i++) {
22         for (j = 0; j < 4; j++) {
23             p[i][j] = L * i + j + 1;
24         }
25     }
26
27     afiseazaMatrice(p,L,C);
28
29     return 0;
30 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc exempluTransmitere_static.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
1 2 3 4
4 5 6 7
7 8 9 10
```

# Transmiterea tablourilor 2D ca argumente funcților

## Alocarea statică (pe STIVĂ)

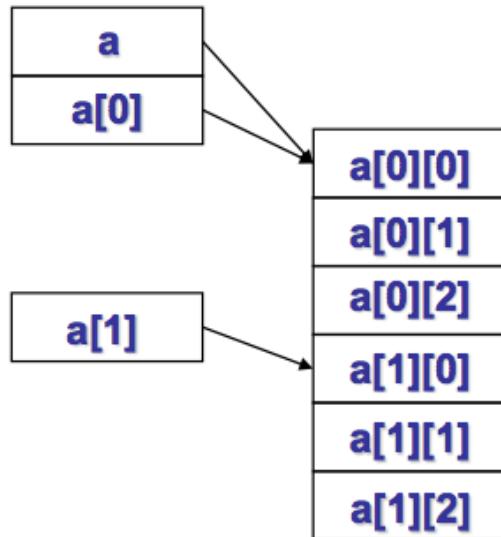
```
4 void afiseazaMatrice(int x[][], int nrLinii, int nrColoane)
5 {
6     int i,j;
7     for (i = 0; i < nrLinii; i++) {
8         for (j = 0;j < nrColoane; j++) {
9             printf("%d ", x[i][j]);
10        }
11        printf("\n");
12    }
13 }
```

```
exempluTransmitere_static.c:4:27: error: array has incomplete element type
      'int []'
void afiseazaMatrice(int x[][], int nrLinii, int nrColoane)
1 error generated.
```

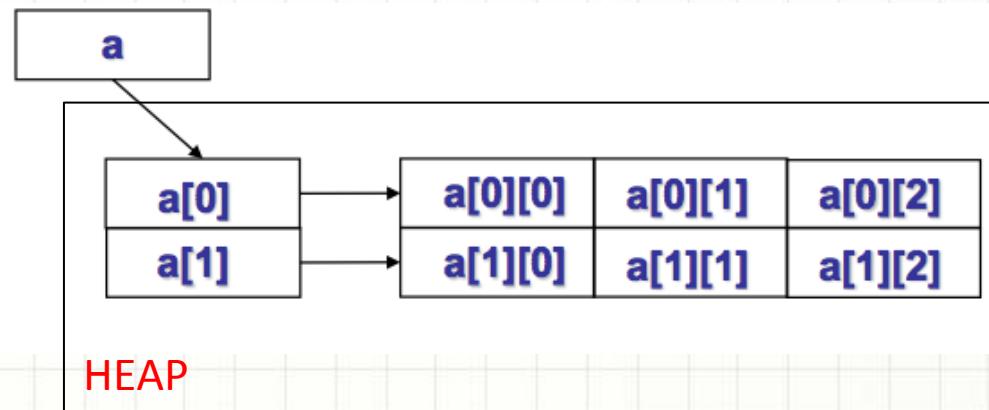
# Transmiterea tablourilor 2D ca argumente funcțiilor

## Alocarea statică (pe STIVĂ)

```
int a[2][3];
```



## Alocarea dinamică (pe HEAP)

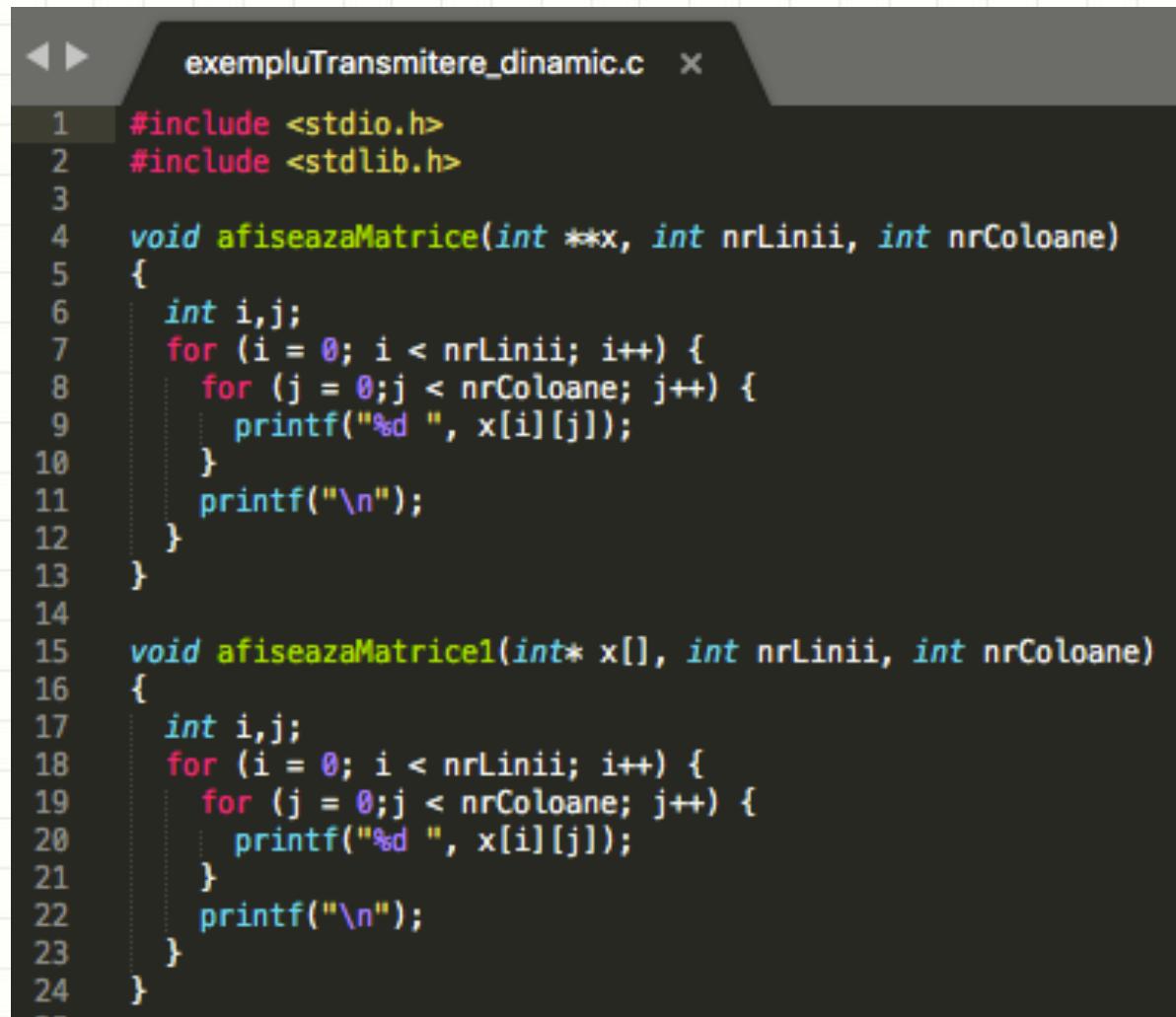


`a` e pointer dublu

- pentru tablouri 2D în alocare dinamică:

- trebuie să transmit pointerul dublu dar am nevoie de numărul de linii și de coloane;
  - alternativ transmit un vector de pointeri și am nevoie de lungimea vectorului (numărul de linii) iar apoi de lungimea fiecărei linii (numărul de coloane);

# Transmiterea tablourilor 2D ca argumente funcțiilor



The image shows a screenshot of a code editor with a dark theme. The file being edited is named "exempluTransmitere\_dinamic.c". The code contains two functions: "afiseazaMatrice" and "afiseazaMatrice1". Both functions print a 2D matrix to the console using nested loops and the printf function. The first function uses a double pointer to the matrix, while the second uses a single pointer.

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

void afiseazaMatrice(int **x, int nrLinii, int nrColoane)
{
    int i,j;
    for (i = 0; i < nrLinii; i++) {
        for (j = 0;j < nrColoane; j++) {
            printf("%d ", x[i][j]);
        }
        printf("\n");
    }
}

void afiseazaMatrice1(int* x[], int nrLinii, int nrColoane)
{
    int i,j;
    for (i = 0; i < nrLinii; i++) {
        for (j = 0;j < nrColoane; j++) {
            printf("%d ", x[i][j]);
        }
        printf("\n");
    }
}
```

# Transmiterea tablourilor 2D ca argumente funcțiilor

```
27 int main() {
28     int L, C, i, j;
29     int **p; // Adresa matrice
30
31     L = 3, C = 4;
32     p = (int**) malloc(L * sizeof(int*));
33
34     for (i = 0; i < L; i++)
35     {
36         p[i] = calloc(C, sizeof(int));
37         printf("Linia %d incepe la %p \n", i, p[i]);
38     }
39
40     for (i = 0; i < L; i++) {
41         for (j = 0; j < C; j++) {
42             p[i][j] = L * i + j + 1;
43             printf("Adresa lui p[%d][%d] este = %p \n", i, j, &p[i][j]);
44         }
45     }
46
47     afiseazaMatrice(p,L,C);
48     afiseazaMatrice1(p,L,C);
49
50
51     return 0;
52 }
```

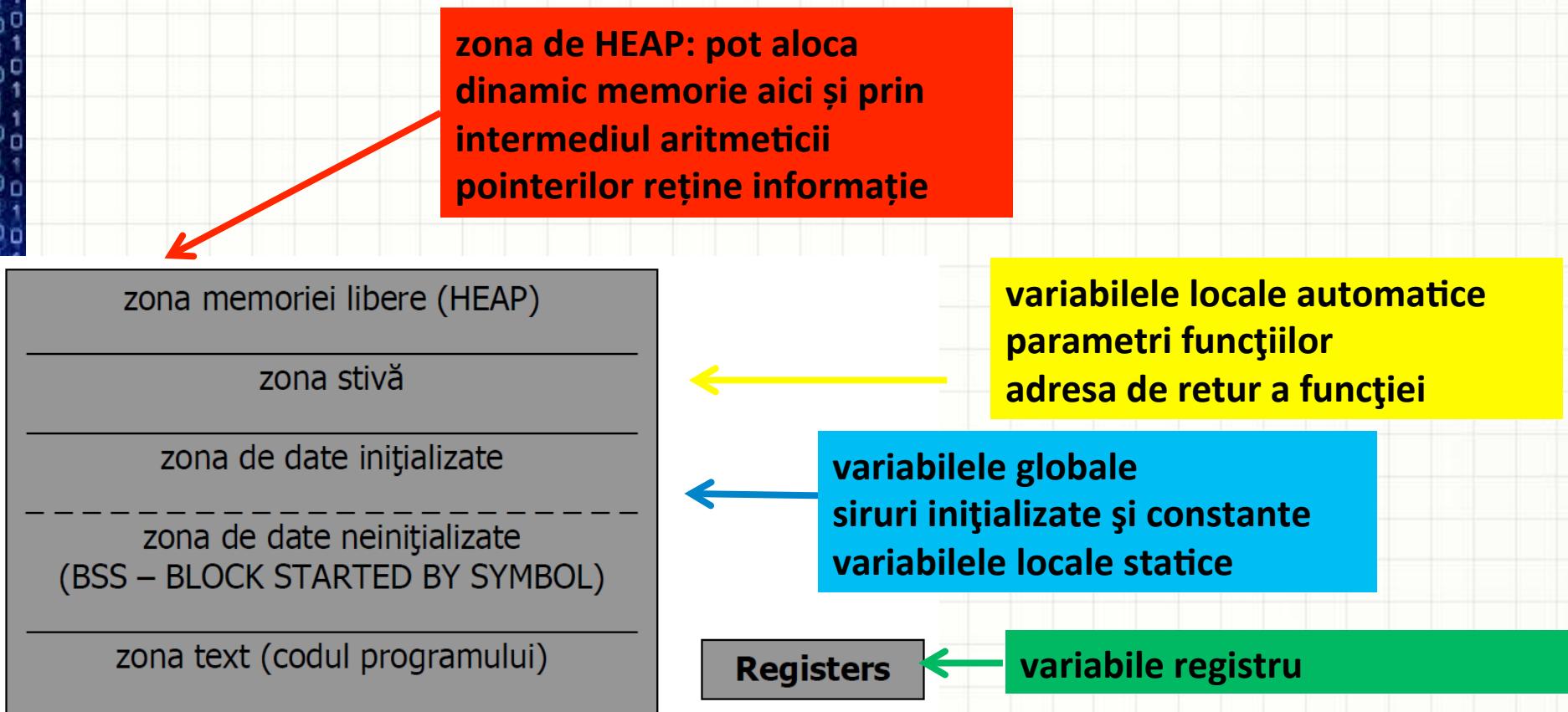
# Transmiterea tablourilor 2D ca argumente funcțiilor

```
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ gcc exempluTransmitere_dinamic.c
[Bogdan-Alexes-MacBook-Pro:curs8 bogdan$ ./a.out
Linia 0 incepe la 0x7faf79d00020
Linia 1 incepe la 0x7faf79d00030
Linia 2 incepe la 0x7faf79d00040
Adresa lui p[0][0] este = 0x7faf79d00020
Adresa lui p[0][1] este = 0x7faf79d00024
Adresa lui p[0][2] este = 0x7faf79d00028
Adresa lui p[0][3] este = 0x7faf79d0002c
Adresa lui p[1][0] este = 0x7faf79d00030
Adresa lui p[1][1] este = 0x7faf79d00034
Adresa lui p[1][2] este = 0x7faf79d00038
Adresa lui p[1][3] este = 0x7faf79d0003c
Adresa lui p[2][0] este = 0x7faf79d00040
Adresa lui p[2][1] este = 0x7faf79d00044
Adresa lui p[2][2] este = 0x7faf79d00048
Adresa lui p[2][3] este = 0x7faf79d0004c
1 2 3 4
4 5 6 7
7 8 9 10
1 2 3 4
4 5 6 7
7 8 9 10
```

# Cuprinsul cursului de azi

1. Alocarea dinamică a memoriei
2. Clase de memorare
3. Siruri de caractere – funcții specifice de manipulare

# Harta simplificată a memoriei la rularea unui program



# Clase de alocare/memorare

- Într-un program C felul în care declarăm variabilele definește modul de alocare al acestora. Clasa de alocare a unei variabile definește următoarele caracteristici:
  - locul în memorie unde se rezervă spațiu pentru variabilă;
  - durata de viață;
  - vizibilitatea;
  - modalitatea de inițializare.
- clase de alocare:
  - **auto(matic)**
  - **register**
  - **static (intern)**
  - **static extern**

# Clasa de alocare auto

- este implicită (variabile locale). Am discutat despre variabile locale în cursurile trecute;
- se specifică prin cuvântul cheie **auto**;
- în mod implicit toate variabilele locale sunt memorate în stivă;
- spațiul de memorie se alocă la execuție;
- variabilele automatice sunt vizibile numai în corpul funcțiilor/instrucțiunilor compuse în care au fost declarate; la revenirea din execuția funcțiilor/instrucțiunilor compuse variabilele se elimină și stiva revine la starea dinaintea apelului;
- nu sunt inițializate;
- parametrii funcțiilor sunt implicit de clasă auto. Ei sunt transmiși, de asemenea, prin stivă (**de la dreapta la stânga!**).

```
int a,b,c;  
auto int d;
```

```
void f(int *x, int *y)  
{    auto int t; t=*x; *x=*y; *y=t; }
```

# Clasa de alocare register

- se specifică prin cuvântul cheie **register**: se cere un acces rapid (registru procesorului) la o variabilă. Nu se garantează că cererea va fi satisfăcută.
- numai parametri și variabilele automatice de tipul **int**, **char** și **pointer** pot fi declarate ca variabile registru;
- **nu există adresă de memorie asociată**;
- nu puteți să manipulați tablouri de regiștri (aveți nevoie la derefențiere de adresa elementului de început al tabloului)
- număr limitat de variabile în regiștri (compilatorul le trece pe cele pe care nu le poate aloca în clasa auto);
- parametri formali pot fi declarați în clasa register.

Exemplu:

```
register int i,s;  
for(i=0;i<n;i++)  
    s = s + i;
```

compilatoarele moderne nu au nevoie de asemenea declarații, ele reușesc să optimizeze codul mai bine decât am putea noi prin declararea variabilelor de tip register

# Clasa de alocare static intern

- se specifică prin cuvântul cheie **static**;
- desemnează variabile cu adrese de memorie constantă, adresa e fixă pe durata executării programului;
- se alocă de compilator în zone speciale (zona de date);
- variabilele globale sunt implicit din clasa static;
- variabilele din clasa static se initializează numai la primul apel.

```
exempluStatic.c
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void adunare(int x)
5 {
6     int static y = 25;
7     y = y + x;
8     printf("y = %d \n", y);
9 }
10
11 int main()
12 {
13     adunare(1);
14     adunare(2);
15     adunare(3);
16     return 0;
17 }
```

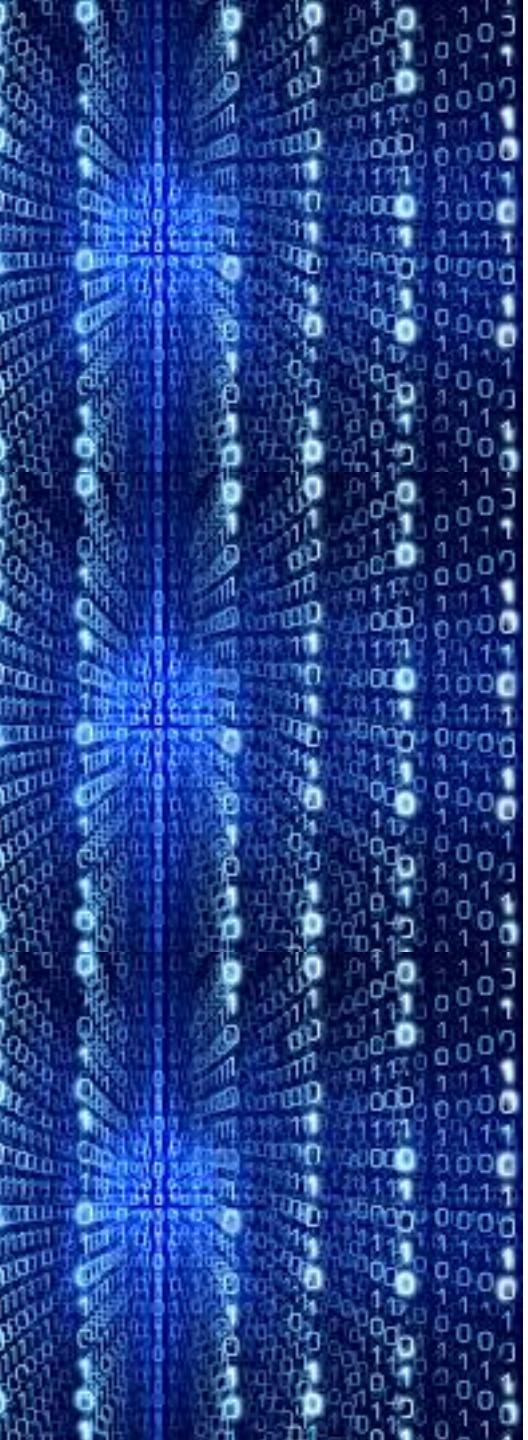
```
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ gcc exempluStatic.c
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ ./a.out
y = 26
y = 28
y = 31
```

# Clasa de alocare static intern

- variabilele din clasa static nu sunt globale, sunt vizibile numai în funcțiile/ blocurile de funcții în care au fost declarate
- exemplu de utilizare: se pot folosi când scriem funcții recursive să numărăm câte apeluri generează o funcție

```
fibonacci.c
x
1 #include<stdio.h>
2
3 int fib(int x)
4 {
5     int static nrApeluri = 0;
6     nrApeluri = nrApeluri + 1;
7     printf("Apelul nr %d \n",nrApeluri);
8     if (x==0 || x== 1)
9         return x;
10    return fib(x-1) + fib(x-2);
11 }
12
13 int main()
14 {
15     fib(20);
16     return 0;
17 }
```

```
Apelul nr 21881
Apelul nr 21882
Apelul nr 21883
Apelul nr 21884
Apelul nr 21885
Apelul nr 21886
Apelul nr 21887
Apelul nr 21888
Apelul nr 21889
Apelul nr 21890
Apelul nr 21891
Bordan-Alexes-MacBook-Pro:cursa9 bordan'
```



# PROGRAMARE PROCEDURALĂ

Bogdan Alexe

[bogdan.alexe@fmi.unibuc.ro](mailto:bogdan.alexe@fmi.unibuc.ro)

Secția Informatică, anul I,

2018-2019

Cursul 10

# 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, siruri 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.
  - Fișiere binare**
    - Funcții specifice de manipulare.
  - Funcții (1)**
    - Declarare și definire. Apel. Metode de transmitere a parametrilor. Pointeri la funcții.
  - Tablouri și pointeri**
    - Aritmetică pointerilor
    - Legătura dintre tablouri și pointeri
    - Alocarea dinamică a memoriei
    - Clase de memorare
  - Siruri de caractere**
    - 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**
- 

# Cuprinsul cursului de azi

1. Clase de memorare
2. Siruri de caractere – funcții specifice de manipulare

# Clase de alocare/memorare

- Într-un program C felul în care declarăm variabilele definește modul de alocare al acestora. Clasa de alocare a unei variabile definește următoarele caracteristici:
  - locul în memorie unde se rezervă spațiu pentru variabilă;
  - durata de viață;
  - vizibilitatea;
  - modalitatea de inițializare.
- clase de alocare:
  - **auto(matic)**
  - **register**
  - **static (intern)**
  - **static extern**

# Clasa de alocare static extern

- ❑ se specifică prin cuvântul cheie **extern**;
- ❑ o variabilă de tip extern este o variabilă definită într-un alt fișier sursă (extern);
- ❑ se alocă în funcție de modul de declarare din fișierul sursă.

Exemplu:

//fisier1.c

**extern int i; //declara variabila i ca fiind definită in alt fisier**

//fisier2.c

**int i = 5; //variabila i este definită aici**

O variabilă poate fi declarată în mai multe fișiere (cu extern), dar trebuie definită într-un singur fișier!

# Clasa de alocare static extern

```
exempluExtern1.c
```

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

```
exempluExtern2.c
```

```
1 #include<stdio.h>
2
3 int x = 1;
4
5 void f()
6 {
7     int x = 7;
8     {
9         extern int x;
10         printf("%d \n",x);
11     }
12 }
13
14 int main()
15 {
16     f();
17     return 0;
18 }
```

Afiseaza 7. Vreau sa am acces la variabila globala x = 1. Cum fac?

# Cuprinsul cursului de azi

1. Clase de memorare
2. Siruri de caractere – funcții specifice de manipulare

# Şiruri de caractere

- **un sir de caractere (string)** este un tablou unidimensional cu elemente de tip char terminat cu caracterul '\0' (NUL)
- o zonă de memorie ocupată cu caractere (un caracter 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 la primul octet. Se poate reprezenta ca:
  - tablou de caractere (pointer constant):
    - `char sir1[10]; //se aloca 10 octeti`
    - `char sir2[10] = "sir2"; //se aloca 10 octeti`
    - `char sir3[] = "sir3"; //se aloca 5 octeti (se mai adauga '\0')`
  - pointer la caractere:
    - `char *sir4; //se aloca memorie numai pentru pointer`
    - `char *sir5 = "sir5"; //se aloca 8 octeti pentru pointer`

# Şiruri de caractere

```
00 ◀ ▶ exempluSiruri.c ×
01
02 1 #include<stdio.h>
03 2 #include<string.h>
04
05 3
06 4 void afiseazaSir(char *t)
07 5 {
08 6     printf("Sirul %s incepe la adresa %p si are dimensiunea in memorie %lu \n",t,t,sizeof(t));
09 7 }
10
11 8
12 9 int main()
13 10 {
14 11
15 12     char sir1[10] = "sir1";
16 13     char sir2[10] = "sir2";
17 14     char sir3[] = "sir3";
18 15     char *sir4 = "sir4";
19
20 21     printf("Sirul %s incepe la adresa %p si are dimensiunea in memorie %lu \n",sir1,sir1,sizeof(sir1));
21 22     afiseazaSir(sir1);
22
23 23
24 24     printf("Sirul %s incepe la adresa %p si are dimensiunea in memorie %lu \n",sir2,sir2,sizeof(sir2));
25 25     printf("Sirul %s incepe la adresa %p si are dimensiunea in memorie %lu \n",sir3,sir3,sizeof(sir3));
26 26     printf("Sirul %s incepe la adresa %p si are dimensiunea in memorie %lu \n",sir4,sir4,sizeof(sir4));
27
28 28
29 29     return 0;
30 30 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ gcc exempluSiruri.c
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ ./a.out
Sirul sir1 incepe la adresa 0x7fff56940b3e si are dimensiunea in memorie 10
Sirul sir1 incepe la adresa 0x7fff56940b3e si are dimensiunea in memorie 8
Sirul sir2 incepe la adresa 0x7fff56940b34 si are dimensiunea in memorie 10
Sirul sir3 incepe la adresa 0x7fff56940b2b si are dimensiunea in memorie 5
Sirul sir4 incepe la adresa 0x1092bff53 si are dimensiunea in memorie 8
```

# 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` (în loc de `gets`)
  - `char *fgets(char *s, int size, FILE *stream)`
  - `fgets(buffer, sizeof(buffer), stdin);`
  - citește și spațiile

## □ afișare:

- funcția `printf` cu modelatorul de format `%s`;
- funcția `puts` (trece pe linia următoare).

# Citirea și afișarea sirurilor de caractere

## □ exemplu

```
001 // File: exempluSiruri2.c
002
003 #include<stdio.h>
004 #include<string.h>
005
006 int main()
007 {
008     char sir1[] = {'r','a','t','o','n','\0'};
009     char sir2[] = "raton";
010     printf("%s %s \n", sir1, sir2);
011
012     char *sir3 = sir1;
013     sir3[0] = 'b';
014     printf("%s %s %s\n", sir1, sir2, sir3);
015
016     char sir4[10] = "raton";
017     printf("%s \n", sir4);
018
019     for(int i = 0; i < 10; i++)
020         printf("Caracterul %c = codul ASCII %d \n", sir4[i], sir4[i]);
021     sir4[5] = 'i';
022     printf("%s \n", sir4);
023
024     char sir5[10] = "raton";
025     sir5[4] = 0;
026     printf("%s \n", sir5);
027     sir5[3] = '\0';
028     printf("%s \n", sir5);
029
030 }
```

# Citirea și afișarea sirurilor de caractere

## □ exemplu

```
00 exempluSiruri2.c
01
02 1 #include<stdio.h>
03 2 #include<string.h>
04
05 3
06 4 int main()
07 5 {
08 6     char sir1[] = {'r','a','t','o','\n','\0'};
09 7     char sir2[] = "raton";
10 8     printf("%s %s \n", sir1, sir2);
11 9
12 10    char *sir3 = sir1;
13 11    sir3[0] = 'b';
14 12    printf("%s %s %s\n",sir1, sir2,
15 13        sir3);
16 14
17 15    char sir4[10] = "raton";
18 16    printf("%s \n", sir4);
19 20
20 21    for(int i = 0; i < 10; i++)
21 22        printf("Caracterul %c = codul ASCII %d \n",
22 23        sir4[i], sir4[i]);
23 24
24 25    sir5[4] = '\0';
25 26    printf("%s \n", sir5);
26 27
27 28    sir5[3] = '\0';
28 29    printf("%s \n", sir5);
29 30
30 }
```

Bogdan-Alexes-MacBook-Pro:curs9 bogdan\$ gcc exempluSiruri2.c  
Bogdan-Alexes-MacBook-Pro:curs9 bogdan\$ ./a.out  
raton raton  
baton raton baton  
raton  
Caracterul r = codul ASCII 114  
Caracterul a = codul ASCII 97  
Caracterul t = codul ASCII 116  
Caracterul o = codul ASCII 111  
Caracterul n = codul ASCII 110  
Caracterul = codul ASCII 0  
ratoni  
rato  
rat

# Functii predefinite pentru manipularea sirurilor de caractere

- funcții de procesare a sirurilor de caractere specifice incluse în fișierul string.h
- lungimea unui sir – funcția **strlen**
  - antet: int **strlen(const char \*sir)**

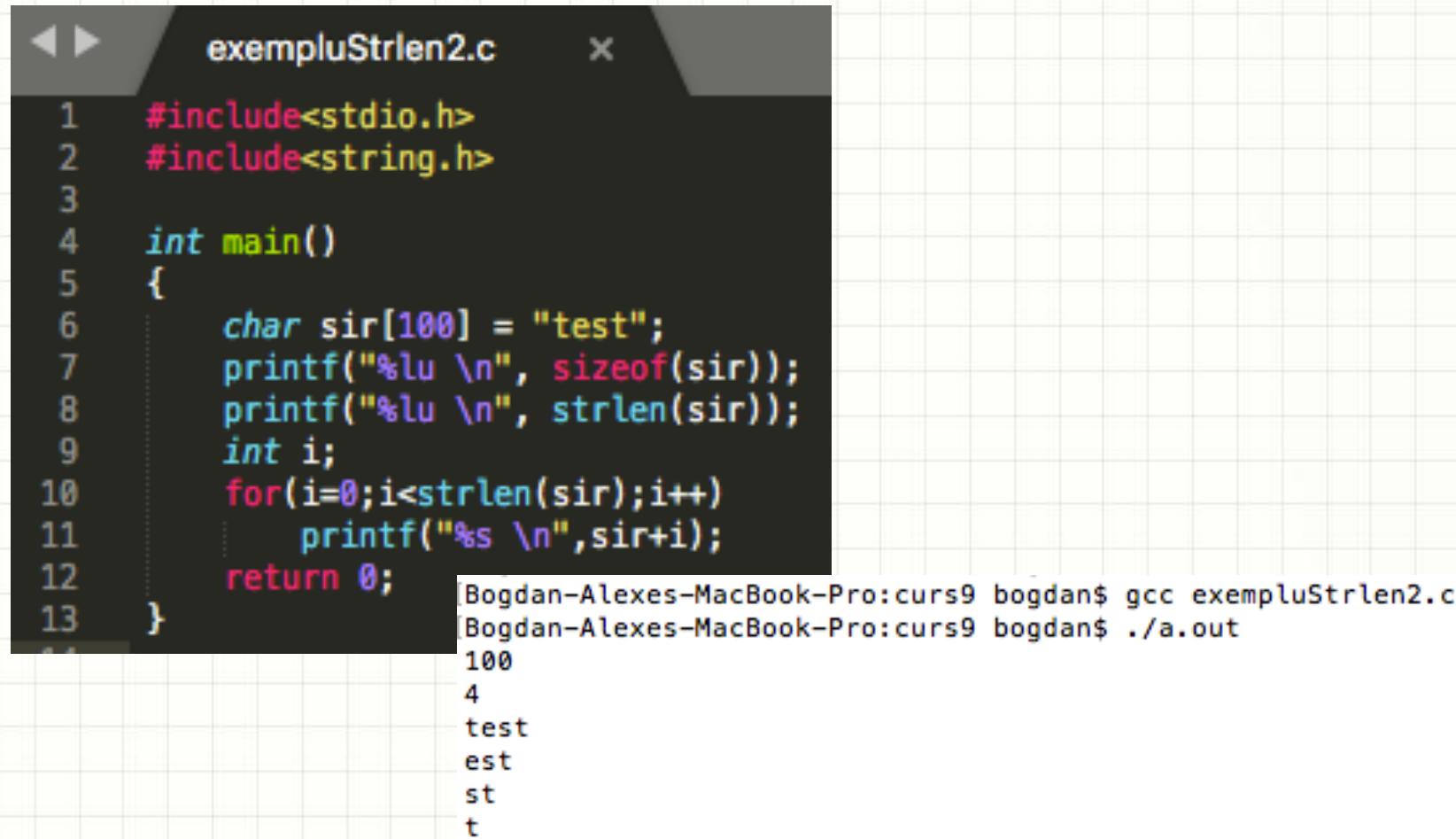
```
exempluStrlen1.c      x

1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char sir[100] = "test";
7     printf("%lu \n", sizeof(sir));
8     printf("%lu \n", strlen(sir));
9     return 0;
10 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ gcc exempluStrlen1.c
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ ./a.out
100
4
```

# Functii predefinite pentru manipularea sirurilor de caractere

- lungimea unui sir – funcția **strlen**
  - antet: **int strlen(const char \*sir)**



```
exempluStrlen2.c      x
1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char sir[100] = "test";
7     printf("%lu \n", sizeof(sir));
8     printf("%lu \n", strlen(sir));
9     int i;
10    for(i=0;i<strlen(sir);i++)
11        printf("%s \n",sir+i);
12    return 0;
13 }
```

Bogdan-Alexes-MacBook-Pro:curs9 bogdan\$ gcc exempluStrlen2.c  
Bogdan-Alexes-MacBook-Pro:curs9 bogdan\$ ./a.out  
100  
4  
test  
est  
st  
t

# Funcții predefinite pentru manipularea sirurilor de caractere

- copierea unui sir
  - nu se poate copia conținutul unui sir în alt sir folosind operația de atribuire (se copiază pointerul și nu conținutul).

```
exempluSiruri3.c
```

```
1 #include<stdio.h>
2 #include<string.h>
3
4
5 int main()
6 {
7     char sir6[10] = "raton";
8     char *sir7 = "baton";//nu am voie sa fac sir7[0] = 'r';
9
10    printf("Adresa lui sir6 este %p \n", sir6);
11    printf("Adresa lui sir7 este %p \n", sir7);
12
13    sir7 = sir6;
14    puts(sir7);
15
16    printf("Adresa lui sir6 este %p \n", sir6);
17    printf("Adresa lui sir7 este %p \n", sir7);
18
19
20
21 }
```

# Functii predefinite pentru manipularea sirurilor de caractere

- copierea unui sir
  - nu se poate copia continutul unui sir în alt sir folosind operația de atribuire (se copiază pointerul și nu continutul).

```
exempluSiruri3.c
```

```
00 1 #include<stdio.h>
01 2 #include<string.h>
02 
03 
04 5 int main()
05 {
06     char sir6[10] = "raton";
07     char *sir7 = "baton";//nu am voie sa fac sir7[0] = 'r';
08 
09     printf("Adresa lui sir6 este %p \n", sir6);
10    printf("Adresa lui sir7 este %p \n", sir7);
11 
12    sir7 = sir6;
13    puts(sir7);
14 
15    printf("Adresa lui sir6 este %p \n", sir6);
16    printf("Adresa lui sir7 este %p \n", sir7);
17    return 0;
18 }
```

```
Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ gcc exempluSiruri3.c
Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ ./a.out
Adresa lui sir6 este 0x7fff52096b3e
Adresa lui sir7 este 0x10db69f74
raton
Adresa lui sir6 este 0x7fff52096b3e
Adresa lui sir7 este 0x7fff52096b3e
```

# Funcții predefinite pentru manipularea sirurilor de caractere

- copierea unui sir
  - nu se poate copia conținutul unui sir în alt sir folosind operația de atribuire (se copiază pointerul și nu conținutul).

```
exempluSiruri4.c
```

```
1 #include<stdio.h>
2 #include<string.h>
3
4
5 int main()
6 {
7     char sir6[10] = "raton";
8     char sir7[10] = "baton";//acum am voie sa fac sir7[0] = 'r';
9
10    printf("Adresa lui sir6 este %p \n", sir6);
11    printf("Adresa lui sir7 este %p \n", sir7);
12
13    sir7 = sir6;
14    puts(sir7);
15
16    printf("Adresa lui sir6 este %p \n", sir6);
17    printf("Adresa lui sir7 este %p \n", sir7);
18
19    return 0;
20 }
```

# Functii predefinite pentru manipularea sirurilor de caractere

- copierea unui sir
  - nu se poate copia continutul unui sir în alt sir folosind operația de atribuire (se copiază pointerul și nu continutul).

The screenshot shows a terminal window with the title "exempluSiruri4.c". The code in the terminal is:

```
1 #include<stdio.h>
2 #include<string.h>
3
4
5 int main()
6 {
7     char sir6[10] = "raton";
8     char sir7[10] = "baton";//acum am voie sa fac sir7[0] = 'r';
9
10    printf("Adresa lui sir6 este %p \n", sir6);
11    printf("Adresa lui sir7 este %p \n", sir7);
12
13    sir7 = sir6;
14    puts(sir7);
15
16    printf("Adresa lui sir6 este %p \n", sir6);
17    printf("Adresa lui sir7 este %p \n", sir7);
18
19    return 0;
20 }
```

At the bottom of the terminal, the output is:

```
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ gcc exempluSiruri4.c
exempluSiruri4.c:15:10: error: array type 'char [10]' is not assignable
          sir7 = sir6;
          ^~~~~~
1 error generated.
```

sir7 este numele unui tablou (pointer constant). Instructiunea sir7=sir6 da eroare la compilare.

# Funcții predefinite pentru manipularea sirurilor de caractere

- copierea unui sir – folosim funcțiile **strcpy** și **strncpy**
  - antet: **char\* strcpy(char \*d, const char\* s);**
    - copiază sirul sursă **s** în sirul destinație **d**;
    - returnează adresa sirului destinație;
    - sirul rezultat are un '\0' la final;
  - antet: **char\* strncpy(char \*d, const char\* s, int n);**
    - copiază primele **n** caractere din sirul sursă **s** în sirul destinație **d**;
    - returnează adresa sirului destinație;
    - sirul rezultatul **NU** are un '\0' la final;

# Funcții predefinite pentru manipularea sirurilor de caractere

- copierea unui sir – folosim funcțiile **strcpy** și **strncpy**



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

int main()
{
    char s[10] = "exemplu";
    char t[10] = "test";
    strncpy(s,t,3);
    printf("%s \n",s);

    s[4] = 0;
    printf("%s \n",s);

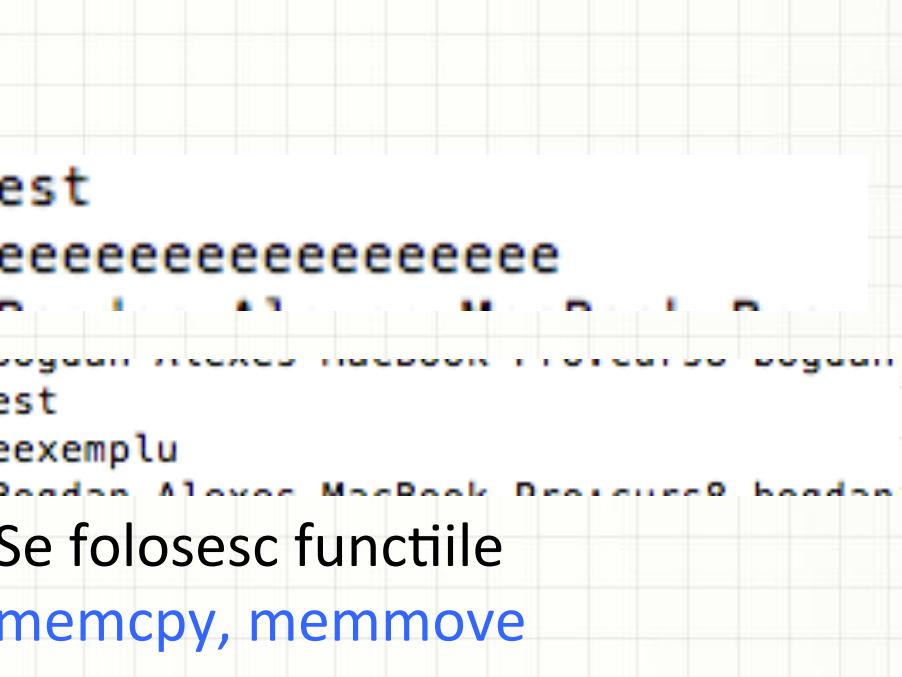
    char p[100] = "nimic";
    strcpy(p,s);
    p[3] = '\0';
    printf("%s \n",p);

    return 0;
}
```

```
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ gcc exempluStrncpy.c
[Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ ./a.out
tesmplu
tesm
tes
```

# Funcții predefinite pentru manipularea sirurilor de caractere

- copierea unui sir – folosim funcțiile **strcpy** și **strncpy**
  - antet: **char\* strcpy(char \*d, const char\* s);**
  - presupune că sirurile destinație și sursa nu se suprapun
    - dacă cele două siruri se suprapun funcția prezintă **undefined behaviour** (comportament nedefinit)



```
exempluStrncpy1.c
1 #include<stdio.h>
2 //#include<string.h>
3
4 int main()
5 {
6     char s[10] = "exemplu";
7     char t[10] = "test";
8
9     strcpy(t,t+1);
10    printf("%s \n",t);
11
12    strcpy(s+1,s);
13    printf("%s \n",s);
14
15    return 0;
16 }
```

est  
eeeeeeeeeeeeeeee  
t e s t  
eexemplu  
Se folosesc functiile  
memcpy, memmove

# Functii predefinite pentru manipularea sirurilor de caractere

- compararea sirurilor – functiile **strcmp** si **strncpy**
  - antet: `int strcmp(const char *s1, const char* s2);`
  - compară lexicografic sirurile s1 și s2;
  - returnează <0 dacă  $s1 <_L s2$ , 0 dacă  $s1 =_L s2$  și >0 dacă  $s1 >_L s2$ ;
  - antet: `int strncpy(const char *s1, const char* s2, int n);`
  - compară lexicografic sirurile s1 și s2 trunchiate la lungimea n
- ambele functii sunt case sensitive
  - `strcmp("POPA","Popa")` returneaza un numar < 0 întrucât 'O' < 'o' (codurile ASCII 79 respectiv 111)
  - unele implementari au functia **stricmp** – case insensitive

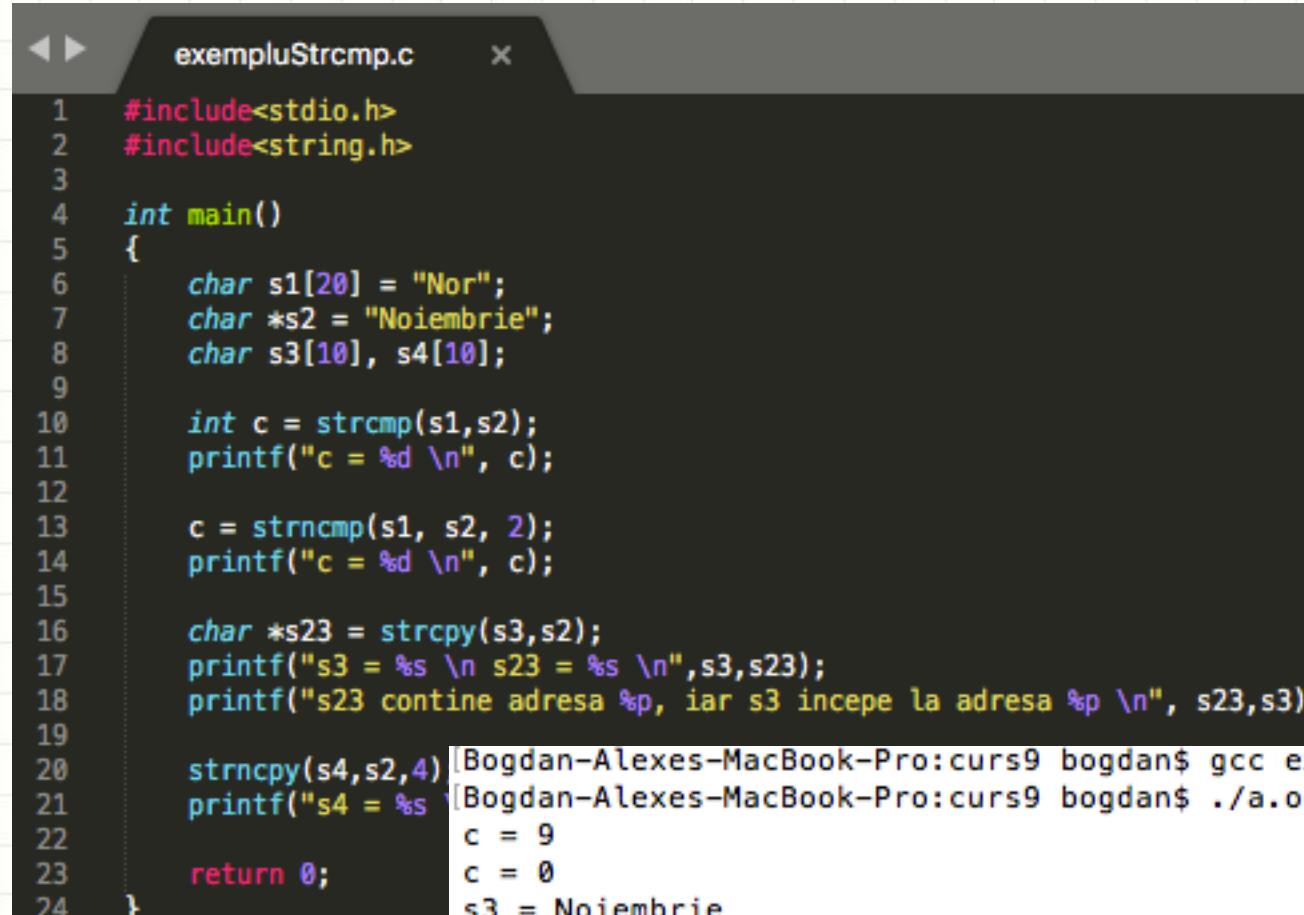
# Functii predefinite pentru manipularea sirurilor de caractere

- compararea sirurilor – functiile **strcmp** și **strncpy**

```
01 exemplustrcmp.c      *
02
03 1 #include<stdio.h>
04 2 #include<string.h>
05
06 3 int main()
07 4 {
08 5     char s1[20] = "Nor";
09 6     char *s2 = "Noiembrie";
10 7     char s3[10], s4[10];
11
12 10    int c = strcmp(s1,s2);
13 11    printf("c = %d \n", c);
14
15 12    c = strncmp(s1, s2, 2);
16 13    printf("c = %d \n", c);
17
18 14    char *s23 = strcpy(s3,s2);
19 15    printf("s3 = %s \n s23 = %s \n",s3,s23);
20 16    printf("s23 contine adresa %p, iar s3 incepe la adresa %p \n", s23,s3);
21
22 17    strncpy(s4,s2,4);
23 18    printf("s4 = %s \n", s4);
24
25 19    return 0;
26 }
```

# Functii predefinite pentru manipularea sirurilor de caractere

- compararea sirurilor – functiile **strcmp** si **strncpy**



```
exempluStrcmp.c      x

1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char s1[20] = "Nor";
7     char *s2 = "Noiembrrie";
8     char s3[10], s4[10];
9
10    int c = strcmp(s1,s2);
11    printf("c = %d \n", c);
12
13    c = strncmp(s1, s2, 2);
14    printf("c = %d \n", c);
15
16    char *s23 = strcpy(s3,s2);
17    printf("s3 = %s \n s23 = %s \n",s3,s23);
18    printf("s23 contine adresa %p, iar s3 incepe la adresa %p \n", s23,s3);
19
20    strncpy(s4,s2,4)
21    printf("s4 = %s \n", s4);
22    c = 9
23    c = 0
24    return 0;
}
```

```
Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ gcc exempluStrcmp.c
Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ ./a.out
c = 9
c = 0
s3 = Noiembrrie
s23 contine adresa 0x7fff57b06b26, iar s3 incepe la adresa 0x7fff57b06b26
s4 = Noie
```

# Funcții predefinite pentru manipularea sirurilor de caractere

- concatenarea sirurilor – funcțiile **strcat** și **strncat**
  - antet: **char\* strcat(char \*d, const char\* s);**
  - concatenează sirul sursă s la sirul destinație d.
  - returnează adresa sirului destinație
  - sirul rezultat are un '\0' la final
  - condiție: sirurile destinație și sursă nu se suprapun, alfel funcția prezintă **undefined behaviour**; (**strcat(s,s) =?**)
  
- antet: **char\* strncat(char \*d, const char\* s, int n);**
- concatenează primele n caractere din sirul sursă s la sirul destinație d
- returnează adresa sirului destinație d
- sirul rezultat **NU** are un '\0' la final

# Funcții predefinite pentru manipularea sirurilor de caractere

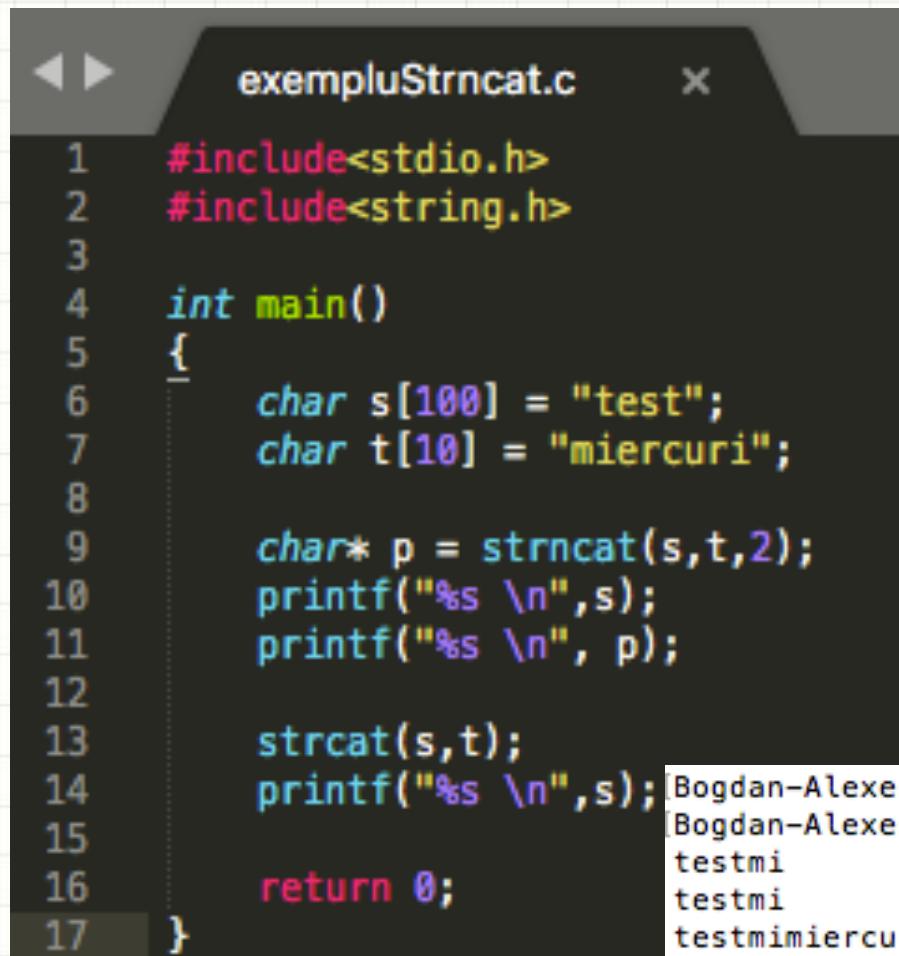
- concatenarea sirurilor – funcțiile **strcat** și **strncat**

```
exempluStrncat.c

1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char s[100] = "test";
7     char t[10] = "miercuri";
8
9     char* p = strncat(s,t,2);
10    printf("%s \n",s);
11    printf("%s \n", p);
12
13    strcat(s,t);
14    printf("%s \n",s);
15
16    return 0;
17 }
```

# Funcții predefinite pentru manipularea sirurilor de caractere

- concatenarea sirurilor – funcțiile **strcat** și **strncat**



```
exempluStrncat.c      x

1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char s[100] = "test";
7     char t[10] = "miercuri";
8
9     char* p = strncat(s,t,2);
10    printf("%s \n",s);
11    printf("%s \n", p);
12
13    strcat(s,t);
14    printf("%s \n",s);
15
16    return 0;
17 }
```

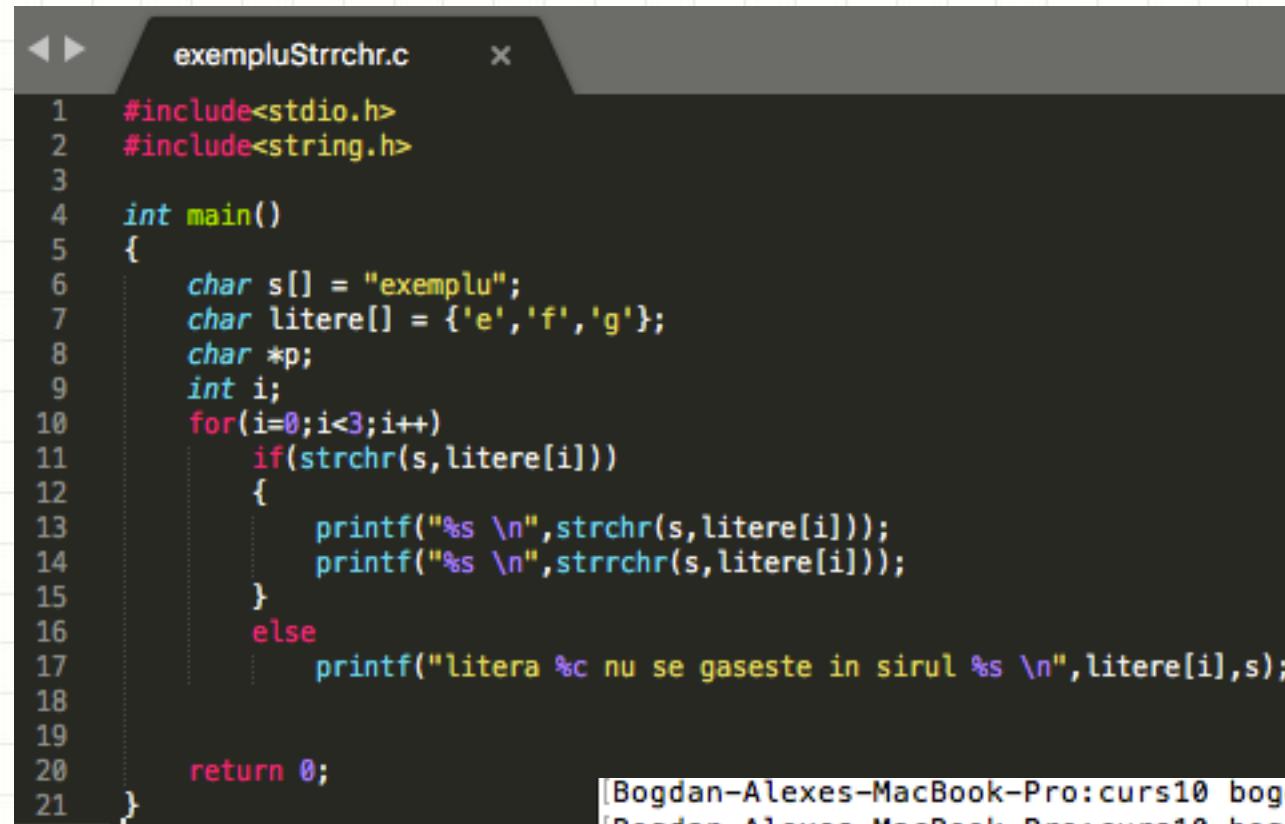
```
Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ gcc exempluStrncat.c
Bogdan-Alexes-MacBook-Pro:curs9 bogdan$ ./a.out
testmi
testmi
testmimiercuri
```

# Funcții predefinite pentru manipularea sirurilor de caractere

- căutarea unui caracter într-un sir – funcțiile **strchr** și **strrchr**
  - antet: `char* strchr(const char *s, char c);`
  - cauță caracterul c în sirul s și întoarce un pointer la prima sa apariție
  - căutare de la stânga la dreapta
  - dacă nu apare caracterul c în sirul s returnează NULL
  
- antet: `: char* strrchr(const char *s, char c);`
- cauță caracterul c în sirul s și întoarce un pointer la prima sa apariție
- căutare de la dreapta la stânga
- dacă nu apare caracterul c în sirul s returnează NULL

# Functii predefinite pentru manipularea sirurilor de caractere

- căutarea unui caracter într-un sir – funcțiile **strchr** și **strrchr**



```
exempluStrrchr.c      x

1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char s[] = "exemplu";
7     char litere[] = {'e','f','g'};
8     char *p;
9     int i;
10    for(i=0;i<3;i++)
11        if(strchr(s,litere[i]))
12        {
13            printf("%s \n",strchr(s,litere[i]));
14            printf("%s \n",strrchr(s,litere[i]));
15        }
16        else
17            printf("litera %c nu se gaseste in sirul %s \n",litere[i],s);
18
19
20    return 0;
21 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluStrrchr.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
exemplu
emplu
litera f nu se gaseste in sirul exemplu
litera g nu se gaseste in sirul exemplu
```

# Funcții predefinite pentru manipularea sirurilor de caractere

- căutarea unui sir în alt sir – funcția **strstr**
  - antet: `char* strstr(const char *s, const char *t);`
  - cauță sirul t în sirul s și întoarce un pointer la prima sa apariție
  - căutare de la stânga la dreapta
  - dacă nu apare sirul t în sirul s returnează NULL
- **exemplu:** să se numere de câte ori apare un sir t într-un sir s.

# Functii predefinite pentru manipularea sirurilor de caractere

- exemplu: să se numere de câte ori apare un sir t într-un sir s.

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

int main()
{
    char s[1000], t[1000];
    printf("Sirul s este : ");scanf("%s",s);
    printf("Sirul t este : ");scanf("%s",t);

    int nrAparitii = 0;
    char *p;

    p = strstr(s,t);

    while(p)
    {
        nrAparitii++;
        p = strstr(p+1,t);
    }

    printf("nrAparitii = %d \n",nrAparitii);

    return 0;
}
```

# Functii predefinite pentru manipularea sirurilor de caractere

- exemplu: să se numere de câte ori apare un sir t într-un sir s.

```
00 exempluStrStr.c  x
01
02 1 #include<stdio.h>
03 2 #include<string.h>
04
05 4 int main()
06 5 {
07 6     char s[1000], t[1000];
08 7     printf("Sirul s este : ");scanf("%s",s);
09 8     printf("Sirul t este : ");scanf("%s",t);
10
11 10     int nrAparitii = 0;
12 11     char *p;
13 12     p = strstr(s,t);
14
15 13     while(p)
16 14     {
17 15         nrAparitii++;
18 16         p = strstr(p+1,t);
19 17     }
20
21 18     printf("nrAparitii = %d \n",nrAparitii);
22 19
23 20     return 0;
24 }
```

```
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluStrStr.c
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
Sirul s este : abracadabra
Sirul t este : ab
nrAparitii = 2
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
Sirul s este : aaaa
Sirul t este : aa
nrAparitii = 3
```

Numără aparițiile suprapuse.  
Dacă vreau aparițiile disjuncte?

# Functii predefinite pentru manipularea sirurilor de caractere

- exemplu: să se numere de câte ori apare un sir t într-un sir s. Număr aparițiile disjuncte.

```
exempluStrStr.c      x
1 #include<stdio.h>
2 #include<string.h>
3
4 int main()
5 {
6     char s[1000], t[1000];
7     printf("Sirul s este : ");scanf("%s",s);
8     printf("Sirul t este : ");scanf("%s",t);
9
10    int nrAparitii = 0;
11    char *p;
12
13    p = strstr(s,t);
14
15    while(p)
16    {
17        nrAparitii++;
18        p = strstr(p+strlen(t),t);
19    }
20
21    printf("nrAparitii = %d \n",nrAparitii);
22
23    return 0;
24 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
Sirul s este : aaaa
Sirul t este : aa
nrAparitii = 2
```

# Funcții predefinite pentru manipularea sirurilor de caractere

- Împărțirea unui sir în subșiruri – funcția **strtok**
  - antet: `char* strtok(char *s, const char *sep);`
  - împarte sirul s în subșiruri conform separatorilor din sirul sep
  - s = “Ana ; are . mere!!”, sep = “ ;,.!?” -> **Ana are mere**
  - string-ul inițial se trimită doar la primul apel al funcției, obținându-se primul subșir
  - la următoarele apeluri, pentru obținerea celorlalte subșiruri se trimit ca prim argument NULL
- exemplu: să se numere cuvintele dintr-o frază

# Functii predefinite pentru manipularea sirurilor de caractere

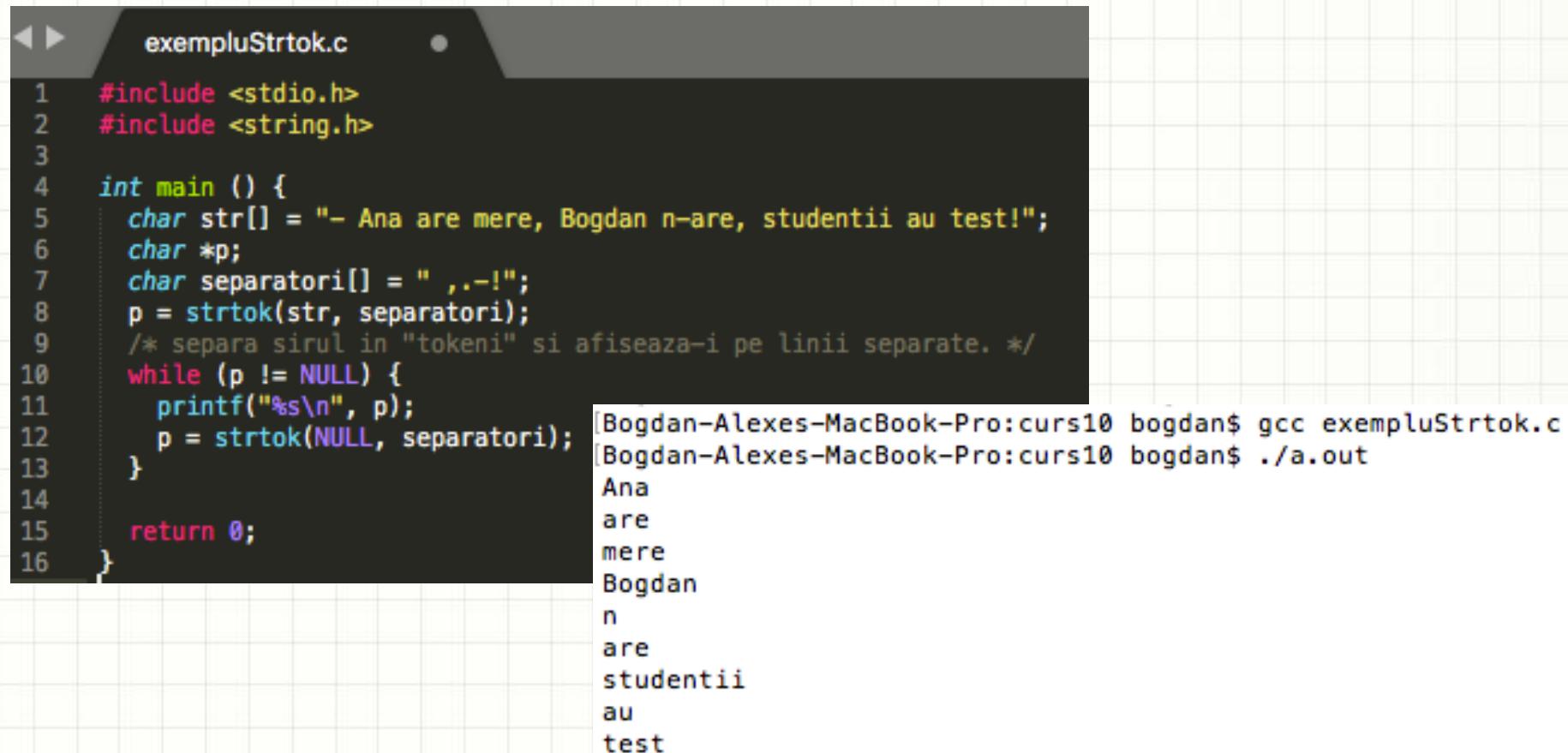
- Împărțirea unui sir în subșiruri – funcția **strtok**
- exemplu: să se numere cuvintele dintr-o frază

```
exempluStrtok.c

1 #include <stdio.h>
2 #include <string.h>
3
4 int main () {
5     char str[] = "- Ana are mere, Bogdan n-are, studentii au test!";
6     char *p;
7     char separatori[] = " ,.-!";
8     p = strtok(str, separatori);
9     /* separa sirul in "tokeni" si afiseaza-i pe linii separate. */
10    while (p != NULL) {
11        printf("%s\n", p);
12        p = strtok(NULL, separatori);
13    }
14
15    return 0;
16 }
```

# Functii predefinite pentru manipularea sirurilor de caractere

- Împărțirea unui sir în subșiruri – funcția **strtok**
- exemplu: să se numere cuvintele dintr-o frază



```
exempluStrtok.c
1 #include <stdio.h>
2 #include <string.h>
3
4 int main () {
5     char str[] = "- Ana are mere, Bogdan n-are, studentii au test!";
6     char *p;
7     char separatori[] = " ,.-!";
8     p = strtok(str, separatori);
9     /* separa sirul in "tokeni" si afiseaza-i pe linii separate. */
10    while (p != NULL) {
11        printf("%s\n", p);
12        p = strtok(NULL, separatori);
13    }
14
15    return 0;
16 }
```

Bogdan-Alexes-MacBook-Pro:curs10 bogdan\$ gcc exempluStrtok.c  
Bogdan-Alexes-MacBook-Pro:curs10 bogdan\$ ./a.out  
Ana  
are  
mere  
Bogdan  
n  
are  
studentii  
au  
test

# Funcții predefinite pentru manipularea sirurilor de caractere

- lungimea maximă a subșirului unui sir sursă **s** ce începe cu primul caracter și e format numai din caractere care apar într-un sir **t** – folosim funcțiile **strspn** și **strcspn**
  - antet: **int strspn(const char \*s, const char\* t);**
    - calculează lungimea maximă a subșirului din **s** ce începe cu primul caracter și e format din caractere care apar în sirul **t**; (**string span**)
    - returnează această lungime
  - antet: **int strcspn(const char \*s,const char\* t);**
    - calculează lungimea maximă subșirului din **s** ce începe cu primul caracter și e format din caractere care NU apar în sirul **t**; (**string complementary span**)
    - returnează această lungime

# Funcții predefinite pentru manipularea sirurilor de caractere

- lungimea maximă a subșirului unui sir sursă **s** ce începe cu primul caracter și e format numai din caractere care apar într-un sir **t** – folosim funcțiile **strspn** și **strcspn**



```
exempluStrspn.c
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     char s[] = "abbcadcfa";
7     char t[] = "bac";
8     printf("%lu \n", strspn(s,t));
9     printf("%lu \n", strcspn(s,t));
10
11     return 0;
12 }
```

```
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluStrspn.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
5
0
```

# Functii predefinite pentru manipularea sirurilor de caractere

- funcțiile **strspn** și **strcspn** sunt folosite la validarea datelor:

```
exempluStrspn.c      x

1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
{
5
6     char cifreNumar[20];
7     int nr;
8     printf("Dati numarul nr = ");scanf("%s",cifreNumar);
9
10    if (strspn(cifreNumar,"0123456789")==strlen(cifreNumar))
11    {
12        sscanf(cifreNumar,"%d",&nr);
13        printf("S-a citit numarul %d\n",nr);
14    }
15    else
16    {
17        printf("Eroare la citirea numarului \n");
18        exit(0);
19    }
20
21    return 0;
22 }
```

Dati numarul nr = 15674

Dati numarul nr = 0098

Dati numarul nr = -100

Dati numarul nr = 10bc34

# Functii predefinite pentru manipularea sirurilor de caractere

- funcțiile **strspn** și **strcspn** sunt folosite la validarea datelor:

```
exemplStrspn2.c    x

1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6
7     char s[20];
8     int nr;
9     printf("Dati sirul s = ");scanf("%s",s);
10
11    if (strspn(s,"aeiouAEIOU")==strlen(s))
12        printf("S-a citit un sir numai din vocale \n");
13
14    if (strcspn(s,"aeiouAEIOU")==strlen(s))
15        printf("S-a citit un sir numai din consoane \n");
16
17    return 0;
18
19 }
```

Dati sirul s = aaaaaeeeiiiiIII00UU  
S-a citit un sir numai din vocale

Dati sirul s = bcdDFTGHH  
S-a citit un sir numai din consoane  
^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^

# Funcții predefinite pentru manipularea sirurilor de caractere

- alte funcții predefinite (mai rar folosite):
  - `char* strpbrk(const char *s, const char* t);`
    - Întoarce adresa subșirului din `s` ce începe cu un caracter care se regăsește în sirul `t`; (`string pointer break`). Dacă nu găsește nici un caracter întoarce `NULL`.
  - `char* strdup(char *s);`
    - Întoarce adresa unei copii în HEAP a sirului `s`
    - `string duplicate`

# Functii predefinite pentru manipularea sirurilor de caractere

The screenshot shows a terminal window with the title "exemplu.c". The code in the terminal is as follows:

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

int main()
{
    char *s="Alina";
    char t[]="Alina";
    char u[30];
    strcpy(u,"Alina");
    printf("%d %d %d\n",sizeof(s),sizeof(t),sizeof(u));
    printf("%d %d %d\n", strlen(s),strlen(t),strlen(u));
    char *ds=(char*)strdup(s);
    ds[3]='\0'; puts(ds); free(ds);
    strcpy(t,"Emil"); puts(t);
    strncpy(u+4,t+1,3); puts(u);
    u[7]='\0'; puts(u);
    strcat(u,s+2); puts(u);
    strncat(u,s,3); puts(u);
    printf("%d %d\n", strcmp(s,"Ali"),strncmp(s,"Ali",3));
    char *fs=strchr("Liliana",'a'); puts(fs);

    return 0;
}
```

The output of the program is:

|               |   |    |
|---------------|---|----|
| 8             | 6 | 30 |
| 5             | 5 | 5  |
| Ali           |   |    |
| Emil          |   |    |
| Alinmil       |   |    |
| Alinmil       |   |    |
| Alinmilina    |   |    |
| AlinmilinaAli |   |    |
| 110           | 0 |    |
| ana           |   |    |
| -             | - | -  |

# Funcții predefinite pentru manipularea sirurilor de caractere

- conversia de la un sir la un număr și invers – funcțiile **sscanf** și **sprintf**
  - conversia de la sir la un număr poate fi făcută cu ajutorul funcției **sscanf** și descriptori de format potriviti
  - exemplu:

```
char *string="-45.8614";
double numar;
sscanf(string, "%lf", &numar);
printf("%f", numar);
```
- conversia de la un număr la sir poate fi făcută cu ajutorul funcției **sprintf** și descriptori de format potriviti
- exemplu:

```
char string[12];
int numar=897645671;
sprintf(string, "%d", numar);
printf("%s", string);
```

# Funcții predefinite pentru manipularea sirurilor de caractere

## □ **int sscanf(char \*sir, const char \*format, adresa1, adresa2, ...)**

unde:

- **sir** este un sir de caractere din care se citesc datele
- **format** este un sir de caractere ce definește textele și formatele datelor care se citesc de la tastatură
- **adresa1, adresa2,...** sunt adresele zonelor din memorie în care se păstrează datele citite după ce au fost convertite

□ funcția **sscanf** întoarce numărul de câmpuri citite și depuse la adresele din listă. Dacă nu s-a stocat nici o valoare, funcția întoarce 0.

□ singura deosebire față de funcția **scanf()** constă în faptul că datele sunt preluate dintr-o zonă de memorie, adresată de primul parametru (și nu de la intrarea standard = stdin).

# Functii predefinite pentru manipularea sirurilor de caractere

- **int sscanf(char \*sir, const char \*format, adresa1, adresa2, ...)**

```
exemplSscanf.c x

1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     int x,y,nr;
7     float f;
8     sscanf("-127","%d",&x);
9     printf("x = %d \n", x);
10    nr = sscanf("-127 -15 3.14","%d %d %f",&x,&y,&f);
11    printf("nr = %d, x = %d, y = %d, f = %f \n",nr,x,y,f);
12    sscanf("12a34","%d",&x);
13    printf("x = %d \n",x);
14
15    return 0;
16 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exemplSscanf.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
x = -127
nr = 3, x = -127, y = -15, f = 3.140000
x = 12
```

# Funcții predefinite pentru manipularea sirurilor de caractere

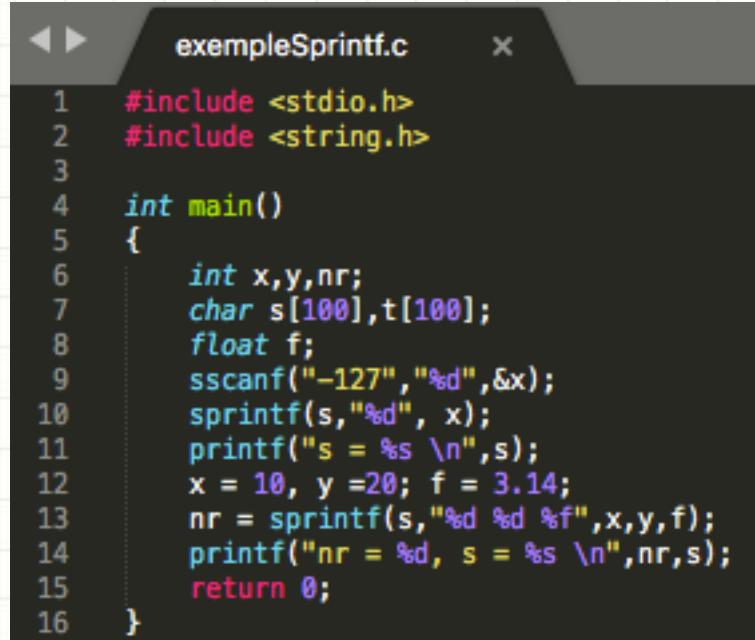
- **int sprintf(char \*sir, const char \*format, arg1, arg2, ...)**

unde:

- **sir** este un sir de caractere in care se scrie
- **format** este un sir de caractere ce definește textele și formatele datelor care se scriu
- **arg1, arg2,...** sunt expresii. Valorile lor se scriu in **sir** conform specificatorilor de format prezenti în format
- funcția **sprintf** întoarce numărul de octeți scriși sau -1 în caz de eșec.
- singura deosebire față de funcția **printf()** constă în faptul că datele sunt scrise într-o zonă de memorie, adresată de primul parametru (și nu la iesirea standard = stdout).

# Functii predefinite pentru manipularea sirurilor de caractere

- **int sprintf(char \*sir, const char \*format, arg1, arg2, ...)**



```
exemplSprintf.c
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     int x,y,nr;
7     char s[100],t[100];
8     float f;
9     scanf("-127",&x);
10    sprintf(s,"%d", x);
11    printf("s = %s \n",s);
12    x = 10, y =20; f = 3.14;
13    nr = sprintf(s,"%d %d %f",x,y,f);
14    printf("nr = %d, s = %s \n",nr,s);
15    return 0;
16 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exemplSprintf.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
s = -127
nr = 14, s = 10 20 3.140000
```

# Funcții predefinite pentru manipularea sirurilor de caractere

- funcții de clasificare (macro-uri) a caracterelor (nu a sirurilor de caractere)
- sunt în fișierul ctype.h

`islower(c)` 1 dacă  $c \in \{‘a’..‘z’\}$ , 0 altfel

`isupper(c)` 1 dacă  $c \in \{‘A’..‘Z’\}$ , 0 altfel

`isalpha(c)` 1 dacă  $c \in \{‘A’..‘Z’\} \cup \{‘a’..‘z’\}$ , 0 altfel

`isdigit(c)` 1 dacă  $c \in \{‘0’..‘9’\}$ , 0 altfel

`isxdigit(c)` 1 dacă  $c \in \{‘0’..‘9’\} \cup \{‘A’..‘F’\} \cup \{‘a’..‘f’\}$ , 0 altfel

`isalnum(c)` 1 dacă `isalpha(c)` || `isdigit(c)`, 0 altfel

`isspace(c)` 1 dacă  $c \in \{‘ ‘, ‘\n’, ‘\t’, ‘\r’, ‘\f’, ‘\v’\}$ , 0 altfel

`isgraph(c)` 1 dacă c este afișabil, fără spațiu, 0 altfel

`isprint(c)` 1 dacă c este afișabil, cu spațiu, 0 altfel

`ispunct(c)` 1 dacă `isgraph(c)` && `isalnum(c)`, 0 altfel

- conversia din literă mare în literă mică și invers se face folosind funcțiile: `tolower(c)` și `toupper(c)`.

# Functii predefinite pentru manipularea sirurilor de caractere

- funcție care convertește un sir de caractere reprezentând un număr întreg, într-o valoare întreagă. Numărul poate avea semn și poate fi precedat de spații albe.

```
exempluCtype.C
```

```
1 #include<stdio.h>
2 #include<string.h>
3 #include<ctype.h>
4
5 int conversie(char *s)
6 { int i, numar, semn;
7     for(i=0; isspace(s[i]); i++);
8
9     semn = (s[i]=='-')?-1:1;
10
11    if(s[i]=='+') || s[i]=='-')
12        i++;
13
14    for(numar=0;isdigit(s[i]);i++)
15        numar=10*numar+(s[i]-'0');
16
17    return semn*numar;
18 }
19
20 int main()
21 {
22     char s[20];
23     strcpy(s, " -123456789");
24     printf("Sirul s = %s e transformat in valoarea = %d\n", s, conversie(s));
25
26     return 0;
27 }
```

Sirul s = -123456789 e transformat in valoarea = -123456789

# Functii predefinite pentru manipularea sirurilor de caractere

- funcție care convertește un întreg într-un sir de caractere pentru baza 10. Întregul poate avea semn.

The terminal window shows the execution of the program:

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluCtype1.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
Sirul s = -123456789]
```

The code editor window shows the following C code:

```
1 // exempluCtype1.C
2
3 #include<stdio.h>
4 #include<string.h>
5 #include<ctype.h>
6
7 void inversare(char[]);
8
9 void conversieIA(int n, char s[]){
10    int j, semn;
11    if((semn=n)<0)
12        n=-n;
13    j=0;
14    do
15        s[j++]=n%10+'0';
16        while ((n/=10)>0);
17        if(semn<0)
18            s[j++]='-';
19        s[j]='\0';
20        inversare(s);
21    void inversare(char s[])
22    { int i,j;
23        char c;
24        for(i=0,j=strlen(s)-1;i<j;i++,j--)
25            c=s[i], s[i]=s[j], s[j]=c;
26    }
27
28 int main()
29 {
30     char s[20];
31     int n = -123456789;
32     conversieIA(n,s);
33     printf("Sirul s = %s \n", s);
34     return 0;
35 }
```

# Functii predefinite pentru manipularea sirurilor de caractere

- funcție care convertește un întreg fără semn într-un sir de caractere în baza 16.

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>

void inversare(char[]);
void conversieI16A(int n, char s[]){
    int j;
    j=0;
    char baza16[] = "0123456789abcdef";
    do { s[j++] = baza16[n%16]; } while ((n/=16)>0);
    s[j]='\0';
    inversare(s);
}

void inversare(char s[])
{ int i,j;
    char c;
    for(i=0,j=strlen(s)-1;i<j;i++,j--)
        c=s[i], s[i]=s[j], s[j]=c;
}

int main()
{
    char s[20];
    int n = 123;
    conversieI16A(n,s);
    printf("Sirul s = %s \n", s);
    return 0;
}
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluCtype2.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
Sirul s = 7b
```

# Funcții predefinite pentru manipularea blocurilor de memorie

- copierea elementelor unui tablou **a** într-un alt tablou **b**:
  - nu se poate face prin atribuire (**b=a**), întrucât **a** și **b** sunt pointeri constanti;
  - copierea se face element cu element folosind instrucțiuni repetitive (for, while);
  - pentru stringuri (tablouri de caractere) avem funcțiile predefinite **strcpy** și **strncpy**:
    - **char\* strcpy(char \*d, const char\* s);**
      - copiază sirul sursă **s** în sirul destinație **d**;
      - returnează adresa sirului destinație
      - sirul rezultat are un '\0' la final
    - **char\* strncpy(char \*d,const char\* s, int n);**
      - copiază primele **n** caractere sirul sursă **s** în sirul destinație **d**;
      - returnează adresa sirului destinație
      - sirul rezultatul **NU** are un '\0' la final

# Funcții predefinite pentru manipularea blocurilor de memorie

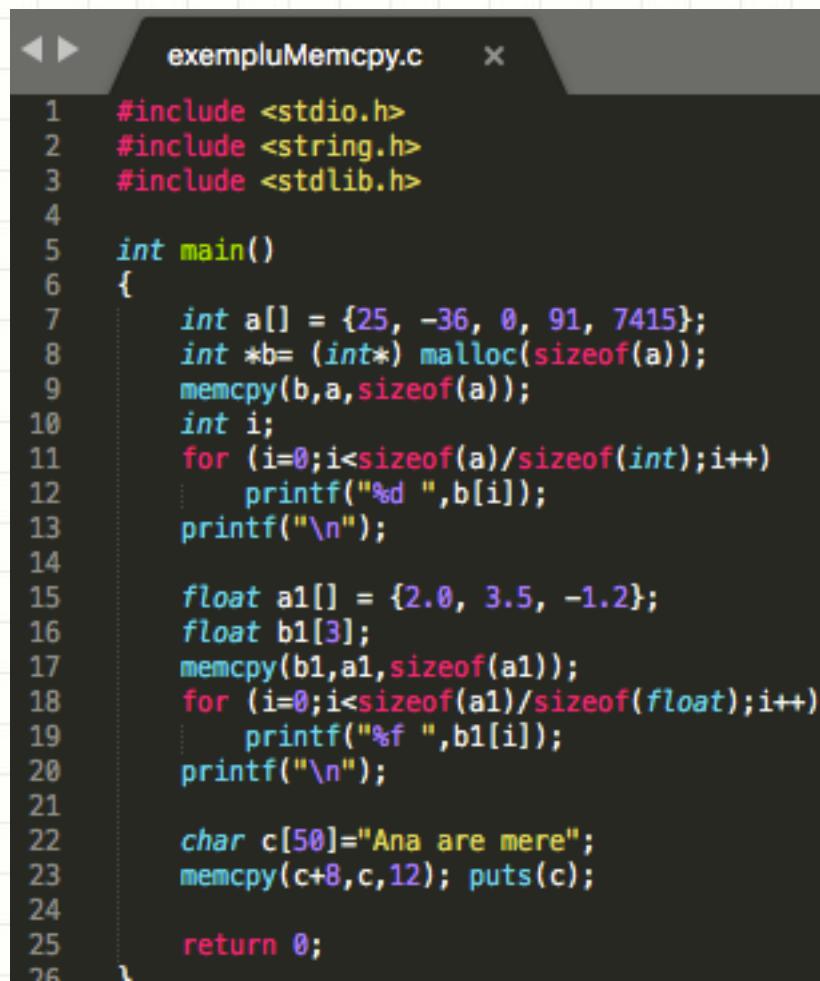
- copierea elementelor unui tablou **a** într-un alt tablou **b**:
  - nu se poate face prin atribuire ( $b=a$ ), întrucât **a** și **b** sunt pointeri constanti;
  - copierea se face element cu element folosind instrucțiuni repetitive (for, while);
  - pe **cazul general** (a și b nu sunt neapărat tablouri de caractere) putem folosi **funcții pentru manipularea blocurilor de memorie: memcpy, memmove;**
    - lucrează la nivel de octet fără semn (unsigned char)
    - alte funcții pentru manipularea blocurilor de memorie: memcmp, memset, memchr

# Funcții predefinite pentru manipularea blocurilor de memorie

- copierea unui tablou – funcțiile **memcpy** și **memmove**
  - antet: **void\* memcpy(void \*d, const void\* s, int n);**
    - copiază primii **n** octeți din sursa **s** în destinația **d**;
    - returnează un pointer la începutul zonei de memorie destinație **d**;
    - un fel de **strcpy** extins (merge și pe alte tipuri de date, nu numai pe char-uri)
    - nu se oprește la octeți = 0 (funcția strcpy se oprește la octeți ce au valoarea 0 = sfârșit de string);
  - presupune că sirurile destinație și sursa nu se suprapun
    - dacă cele două siruri se suprapun funcția prezintă **undefined behaviour** (comportament nedefinit)

# Functii predefinite pentru manipularea blocurilor de memorie

- copierea unui tablou – funcțiile **memcpy** și **memmove**
  - antet: **void\* memcpy(void \*d, const void\* s, int n);**



The screenshot shows a code editor window titled "exempluMemcpy.c". The code demonstrates the use of the `memcpy` function to copy data from one memory location to another. It includes examples for copying integers, floats, and characters.

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

int main()
{
    int a[] = {25, -36, 0, 91, 7415};
    int *b= (int*) malloc(sizeof(a));
    memcpy(b,a,sizeof(a));
    int i;
    for (i=0;i<sizeof(a)/sizeof(int);i++)
        printf("%d ",b[i]);
    printf("\n");

    float a1[] = {2.0, 3.5, -1.2};
    float b1[3];
    memcpy(b1,a1,sizeof(a1));
    for (i=0;i<sizeof(a1)/sizeof(float);i++)
        printf("%f ",b1[i]);
    printf("\n");

    char c[50] = "Ana are mere";
    memcpy(c+8,c,12);
    puts(c);

    return 0;
}
```

# Functii predefinite pentru manipularea blocurilor de memorie

- copierea unui tablou – funcțiile **memcpy** și **memmove**
  - antet: **void\* memcpy(void \*d, const void\* s, int n);**

The screenshot shows a terminal window with two panes. The left pane displays the source code of `exempluMemcpy.c`, and the right pane shows the terminal output.

**Code (Left Pane):**

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

int main()
{
    int a[] = {25, -36, 0, 91, 7415};
    int *b= (int*) malloc(sizeof(a));
    memcpy(b,a,sizeof(a));
    int i;
    for (i=0;i<sizeof(a)/sizeof(int);i++)
        printf("%d ",b[i]);
    printf("\n");

    float a1[] = {2.0, 3.5, -1.2};
    float b1[3];
    memcpy(b1,a1,sizeof(a1));
    for (i=0;i<sizeof(a1)/sizeof(float);i++)
        printf("%f ",b1[i]);
    printf("\n");

    char c[50] = "Ana are mere";
    memcpy(c+8,c,12);
    puts(c);

    return 0;
}
```

**Terminal Output (Right Pane):**

```
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluMemcpy.c
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
25 -36 0 91 7415
2.000000 3.500000 -1.200000
Ana are Ana are Ana
25 -36 0 91 7415
2.000000 3.500000 -1.200000
Ana are Ana are mere
```

# Funcții predefinite pentru manipularea blocurilor de memorie

- copierea unui tablou – funcțiile **memcpy** și **memmove**
  - antet: **void\* memmove(void \*d, const void\* s, int n);**
    - copiază primii **n** octeți din sursa **s** în destinația **d**;
    - returnează un pointer la începutul zonei de memorie destinație **d**;
    - identică cu funcția **memcpy** + tratează cazurile de suprapunere dintre **d** și **s**
  - nu contează că sirurile destinație **d** și sursă **s** se suprapun
    - folosește un buffer intern pentru copiere

# Functii predefinite pentru manipularea blocurilor de memorie

- copierea unui tablou – funcțiile **memcpy** și **memmove**
  - antet: **void\* memmove(void \*d, const void\* s, int n);**

```
exempluMemmove1.c  x

1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 int main()
6 {
7
8     char c[50] = "Ana are mere";
9     memmove(c + 8, c, 12); puts(c);
10
11    char t[] = "memmove este foarte folositor.....";
12    memmove(t + 20, t + 13, 16);
13    puts(t);
14
15    return 0;
16 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluMemmove1.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
Ana are Ana are mere
memmove este foarte foarte folositor.....]
```

# Functii predefinite pentru manipularea blocurilor de memorie

- funcție care elimină toate aparițiile unei siruri dintr-un sir

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

void eliminaAparitii(char *s, char* t)
{
    char *p = strstr(s,t);
    while(p != NULL)
    {
        memmove(p,p+strlen(t),s + strlen(s)- (p + strlen(t)) + 1);
        p = strstr(s,t);
    }
}

int main()
{
    char s[100],t[100];
    strcpy(s,"abbbccca");
    strcpy(t,"bc");
    eliminaAparitii(s,t);
    printf("%s\n",s);
    return 0;
}
```

Nu obțin ceea ce trebuie, unde am greșit?

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluMemmove2.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
aa
```

# Functii predefinite pentru manipularea blocurilor de memorie

- funcție care elimină toate aparițiile unei siruri dintr-un sir

```
exempluMemmove3.c  x

1 #include <stdio.h>
2 #include <string.h>
3
4 void eliminaAparitii(char *s, char* t)
5 {
6     char *p = strstr(s,t);
7     while(p != NULL)
8     {
9         memmove(p,p+strlen(t),s + strlen(s)- (p + strlen(t)) + 1);
10        p = strstr(p,t);
11    }
12 }
13
14 int main()
15 {
16     char s[100],t[100];
17     strcpy(s,"abbbccca");
18     strcpy(t,"bc");
19     eliminaAparitii(s,t);
20     printf("%s\n",s);
21     return 0;
22 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluMemmove3.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
abbcca
```

# Funcții predefinite pentru manipularea blocurilor de memorie

- setarea unor octeți la o valoare – funcția **memset**
  - antet: **void\* memset(void \*d, char val, int n);**
  - În zona de memorie dată de pointerul d, sunt setate primele n poziții (octeți) la valoarea dată de val. Funcția returnează sirul d.



```
exempluMemset.c
1 #include <stdio.h>
2 #include <string.h>
3
4 int main ()
5 {
6     char t[] = "abcdefghijklmnopqrstuvwxyz";
7     memset(t, '-', 8);
8     puts(t);
9     return 0;
10 }
```

```
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluMemset.c
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
-----ijklmnopqrstuvwxyz
```

# Funcții predefinite pentru manipularea blocurilor de memorie

- căutarea unui octet într-un tablou – funcția **memchr**
  - antet: **void\* memchr(const void \*d, char c, int n);**
  - determină prima apariție a octetului **c** în zona de memorie dată de pointerul **d** și care conține **n** octeți. Funcția returnează pointerul la prima apariție a lui **c** în **d** sau **NULL**, dacă **c** nu se găsește în **d**.

```
exempluMemchr.c  x
1 #include <stdio.h>
2 #include <string.h>
3
4 int main ()
5 {
6     char t[] = "abcdefghijklmnopqrstuvwxyz";
7     char *p = memchr(t, 'm', 25);
8     puts(p);
9
10    return 0;
11 }
```

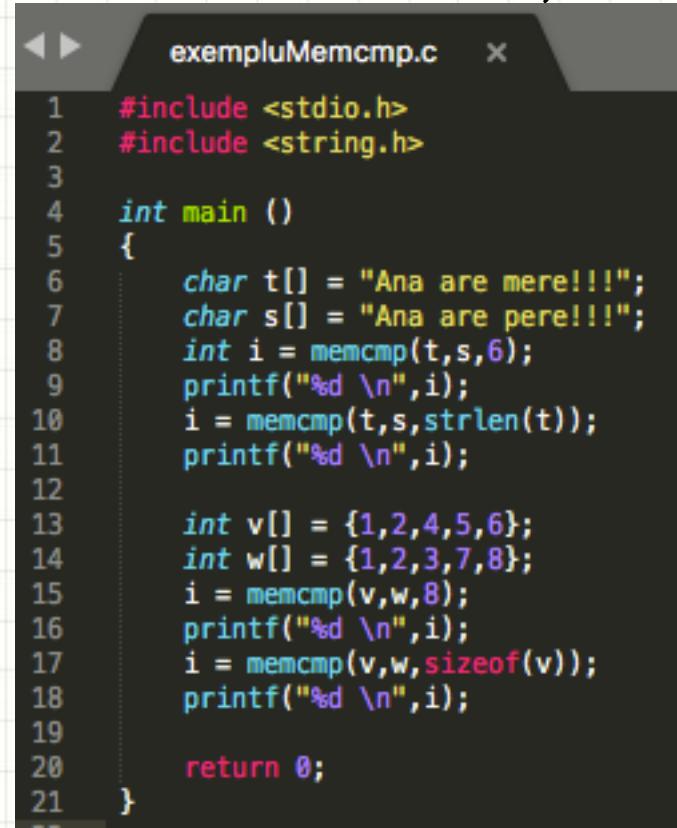
```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluMemchr.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
mnopqrstuvwxyz
```

# Funcții predefinite pentru manipularea blocurilor de memorie

- compararea a două tablouri pe octeți – funcția **memcmp**
  - antet: `int memcmp(const void *s1, const void * s2, int n);`
  - compara primii **n** octeți corespondenți începând de la adresele **s1** și **s2**.
  - returnează 0 dacă octetii sunt identici, ceva mai mic decât 0 dacă **s1 <sub>L</sub> s2**, ceva mai mare decât 0 dacă **s1 >sub>L</sub> s2**

# Functii predefinite pentru manipularea blocurilor de memorie

- compararea a două tablouri pe octeți – funcția **memcmp**
  - antet: **int memcmp(const void \*s1, const void \* s2, int n);**
  - compara primii **n** octeți corespondenți începând de la adresele **s1** și **s2**.



```
exempluMemcmp.c  x

1 #include <stdio.h>
2 #include <string.h>
3
4 int main ()
5 {
6     char t[] = "Ana are mere!!!";
7     char s[] = "Ana are pere!!!";
8     int i = memcmp(t,s,6);
9     printf("%d \n",i);
10    i = memcmp(t,s,strlen(t));
11    printf("%d \n",i);
12
13    int v[] = {1,2,4,5,6};
14    int w[] = {1,2,3,7,8};
15    i = memcmp(v,w,8);
16    printf("%d \n",i);
17    i = memcmp(v,w,sizeof(v));
18    printf("%d \n",i);
19
20    return 0;
21 }
```

```
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc exempluMemcmp.c
[Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
0
-3
0
1
```

# Functiile din string.h

|                                                               |                                                                                                                                                   |
|---------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>char* strcpy(char* d,const char* s)</code>              | copiază sirul <b>s</b> în <b>d</b> , inclusiv '\0', întoarce <b>d</b>                                                                             |
| <code>char* strncpy(char* d,const char* s, int n)</code>      | copiază <b>n</b> caractere din sirul <b>s</b> în <b>d</b> , completând eventual cu '\0', întoarce <b>d</b>                                        |
| <code>char* strcat(char* d,const char* s)</code>              | concatenează sirul <b>s</b> la sfârșitul lui <b>d</b> , întoarce <b>d</b>                                                                         |
| <code>char* strncat(char* d,const char* s, int n)</code>      | concatenează cel mult <b>n</b> caractere din sirul <b>s</b> la sfârșitul lui <b>d</b> , completând cu '\0', întoarce <b>d</b>                     |
| <code>int strcmp(const char* d, const char* s)</code>         | compară sirurile <b>d</b> și <b>s</b> , întoarce<br>-1 dacă <b>d</b> < <b>s</b> ,<br>0 dacă <b>d</b> == <b>s</b> și<br>1 dacă <b>d</b> > <b>s</b> |
| <code>int stricmp(const char* d, const char* s)</code>        | compară sirurile <b>d</b> și <b>s</b> (ca și <code>strcmp()</code> ) fără a face distincție între litere mari și mici                             |
| <code>int strncmp(const char* d, const char* s, int n)</code> | similar cu <code>strcmp()</code> , cu deosebirea că se compară cel mult <b>n</b> caractere                                                        |
| <code>char* strchr(const char* d,char c)</code>               | caută caracterul <b>c</b> în sirul <b>d</b> ; întoarce un pointer la prima apariție a lui <b>c</b> în <b>d</b> , sau <b>NULL</b>                  |
| <code>char* strrchr(const char* d,char c)</code>              | întoarce un pointer la ultima apariție a lui <b>c</b> în <b>d</b> , sau <b>NULL</b>                                                               |
| <code>char* strstr(const char* d, const char* s)</code>       | întoarce un pointer la prima apariție a subșirului <b>s</b> în <b>d</b> , sau <b>NULL</b>                                                         |
| <code>char* strpbrk(const char* d, const char* s)</code>      | întoarce un pointer la prima apariție a unui caracter din subșirul <b>s</b> în <b>d</b> , sau <b>NULL</b>                                         |

# Functiile din string.h

|                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>int strspn(const char* d,<br/>           const char* s)</code>       | întoarce lungimea prefixului din <b>d</b> care conține numai caractere din <b>s</b>                                                                                                                                                                                                                                                                                                                                                |
| <code>int strcspn(const char* d,<br/>           const char* s)</code>      | întoarce lungimea prefixului din <b>d</b> care conține numai caractere ce nu apar în <b>s</b>                                                                                                                                                                                                                                                                                                                                      |
| <code>int strlen(const char* s)</code>                                     | întoarce lungimea lui <b>s</b> ('\'0' nu se numără)                                                                                                                                                                                                                                                                                                                                                                                |
| <code>void* memcpy(void* d,<br/>           const void* s,int n)</code>     | copiaza <b>n</b> octeți din <b>s</b> în <b>d</b> ; întoarce <b>d</b>                                                                                                                                                                                                                                                                                                                                                               |
| <code>void* memmove(void* d,<br/>           const void* s,int n)</code>    | ca și <code>memcpy</code> , folosită dacă <b>s</b> și <b>d</b> se întrepătrund                                                                                                                                                                                                                                                                                                                                                     |
| <code>void* memset(void* d,const int c,<br/>           int n)</code>       | copiază caracter <b>c</b> în primele <b>n</b> poziții din <b>d</b>                                                                                                                                                                                                                                                                                                                                                                 |
| <code>int memcmp(const void* d,<br/>           const void* s,int n)</code> | compară zonele adresate de <b>s</b> și <b>d</b>                                                                                                                                                                                                                                                                                                                                                                                    |
| <code>char* strtok(const char* d,<br/>           const char* s)</code>     | caută în <b>d</b> subșirurile delimitate de caracterele din <b>s</b> ; primul apel întoarce un pointer la primul subșir din <b>d</b> care nu conține caractere din <b>s</b> următoarele apeluri se fac cu primul argument <b>NULL</b> , întorcându-se de fiecare dată un pointer la următorul subșir din <b>d</b> ce nu conține caractere din <b>s</b> ; în momentul în care nu mai există subșiruri, funcția întoarce <b>NULL</b> |

# Implementarea diverselor funcții din string.h

- ❑ putem scrie propriile funcții care realizează operații pe șiruri de caractere
- ❑ câteva exemple:
  - ❑ lungimea unui șir -> strlen
  - ❑ copierea unui șir în alt șir -> strcpy
  - ❑ compararea lexicografică a două șiruri -> strcmp
- ❑ implementări cu tablouri și pointeri

# Lungimea unui sir

```
lungimeSir.c      x

1 #include <stdio.h>
2 #include<string.h>
3
4 int lungimeSir1(char *s)
5 {
6     int lungime = 0, i;
7     for(i = 0; s[i]; i++)
8         lungime++;
9     return lungime;
10 }
11
12 int lungimeSir2(char *s)
13 {
14     char *p=s;
15     while (*p)
16         p++;
17     return p-s;
18 }
19
20
21 int main()
22 {
23     char s[1000];
24     strcpy(s,"Azi e miercuri");
25     printf("lungime sir = %d \n",lungimeSir1(s));
26     printf("lungime sir = %d \n",lungimeSir2(s));
27     return 0;
28 }
```

Ce se intampla daca pun  
while (\*p++)?

```
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ gcc lungimeSir.c
Bogdan-Alexes-MacBook-Pro:curs10 bogdan$ ./a.out
lungime sir = 14
lungime sir = 14
```

# Copierea unui sir în alt sir

```
copiereSir.C x

1 #include <stdio.h>
2
3 char* copiazaSir1(char *d, char* s)
4 {
5     int i=0;
6     while (d[i] = s[i])
7         i++;
8     return d;
9 }
10
11 void copiazaSir2(char *d, char* s)
12 {
13     int i;
14     while (*d = *s)
15     {
16         d++;
17         s++;
18     }
19 }
20
21
22 int main()
23 {
24     char s[1000], t[1000];
25     copiazaSir1(s,"Azi e miercuri");
26     copiazaSir1(t,s);
27     printf("Sirul t este: %s \n",t);
28     copiazaSir2(t,s);
29     printf("Sirul t este: %s \n",t);
30     return 0;
31 }
```

```
-----  
Sirul t este: Azi e miercuri  
Sirul t este: Azi e miercuri  
-----
```

# Compararea lexicografică a două siruri

```
comparareSiruri.C x

1 #include <stdio.h>
2 #include <string.h>
3
4 int comparaSiruri1(char* s1, char *s2)
5 {
6     int j;
7     for(j=0; s1[j]==s2[j]; j++)
8         if(s1[j]=='\0')
9             return 0;
10    return s1[j]-s2[j];
11 }
12
13 int comparaSiruri2(char* s1, char *s2)
14 {
15     for(; *s1==*s2; s1++,s2++)
16         if(*s1=='\0')
17             return 0;
18     return *s1-*s2;
19 }
20
21
22 int main()
23 {
24     char s[1000], t[1000];
25     strcpy(s,"Azi e miercuri");
26     strcpy(t,"Azi e joi");
27
28     int cmp = comparaSiruri1(s,t);
29     printf("%d %d \n",comparaSiruri1(s,t),comparaSiruri2(s,t));
30     return 0;
31 }
```

F / CUISS  
3 3