

Bine ați venit la cursul Tehnici de programare - TP



Cursul 1 – Introducere. Matrici

Ș.l. dr. ing. Cătălin Iapă

catalin.iapa@cs.upt.ro

Ne cunoaștem
deja:

Ne cunoaștem
deja:

Curs:



Cătălin Iapă

Ne cunoaștem
deja:

Curs:



Cătălin Iapă

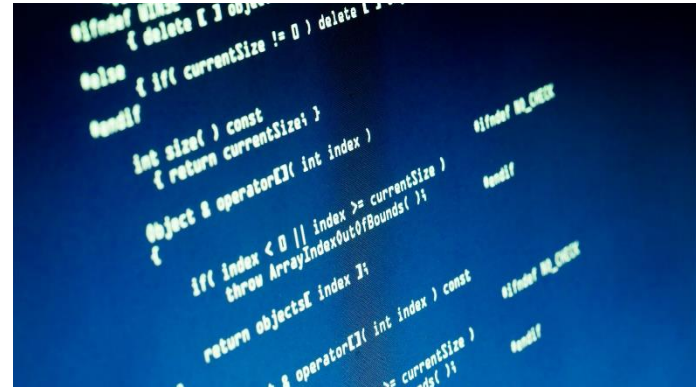
Laboratoare și proiecte:

Beatrice Toader

Alin Stanciu

Detalii administrative curs

- Semestrul 2: 14 săptămâni
 - 2 ore de **curs**/ săptămână
 - 2 ore de **laborator**/ săptămână
 - 2 ore de **proiect**/ 2 săptămâni
- Veți primi **2 note**, media lor este nota finală la LSD
 - 1 notă la examen, în sesiune, după cele 14 săptămâni
 - 1 notă pentru activitatea de la laborator
- Pentru a promova materia e nevoie să luați **cel puțin nota 5**, atât la examen cât și la laborator
- **Examenul** se poate da de 3 ori



Ce facem la TP

Date în memoria calculatorului

Matrici

Matrici alocate dinamic

Exerciții

Despre TP

- Tehnici de Programare
- Ce vom face?
 - Programare în limbajul C
- De ce ai nevoie înainte să începi acest curs?
 - Noțiunile de programare asimilate la materia Programarea Calculatoarelor din Semestrul 1

Despre TP

1. Matrici, inclusiv alocare dinamica matrici
2. Eficienta algoritmilor. Functii de timp si numere aleatoare
3. Pointeri la functii. qsort, bsearch
4. Structuri. Structuri cu campuri pe biti
5. Functii cu numar variabil de argumente
6. Liste simplu inlantuite
7. Liste dublu inlantuite

Despre TP

8. Recursivitate

9. Divide et Impera

10. Greedy

11. Backtraking

12. Tehnici de cautare si sortare



Ce facem la TP

Date în memoria calculatorului

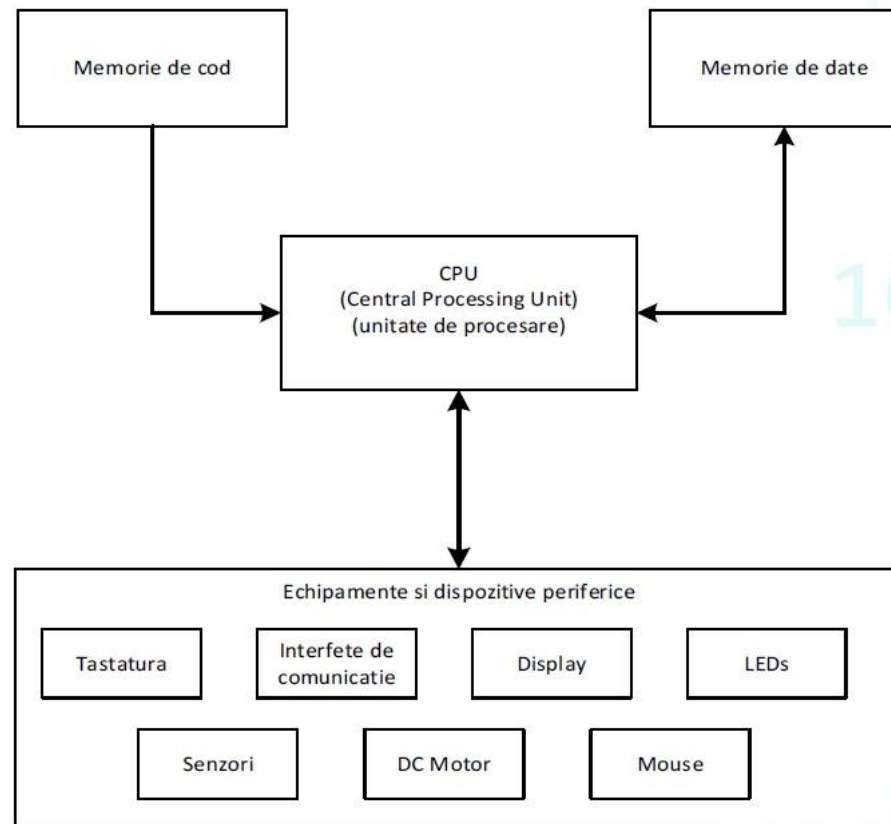
Matrici

Matrici alocate dinamic

Exerciții

Date.

Structura generala a unui sistem de calcul



Date.

- Ce poate reține un calculator în **memorie**?
 - Instrucțiuni
 - Date

Ce fel de date poate stoca un calculator?

- Numere
- Litere (caractere)
- Culori
- Sunete

Date.

- Cum stochează un calculator datele?
 - Sub formă de numere binare

```
00011010100010101110101011110000001010101
01010101000000101111100101010101111000010
00000111100101010101010101010101010101001
01010101010101001010110101111000010101010
```

Numere

123

0111 1011

-7

1000 0111

Litere (caractere) – tabel ASCII

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	@	@	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

Culori

```

#ifdef NDEBUG
    { delete [ ] objects; }
else
    { if( currentSize != 0 ) delete [ ] objects; }
#endif

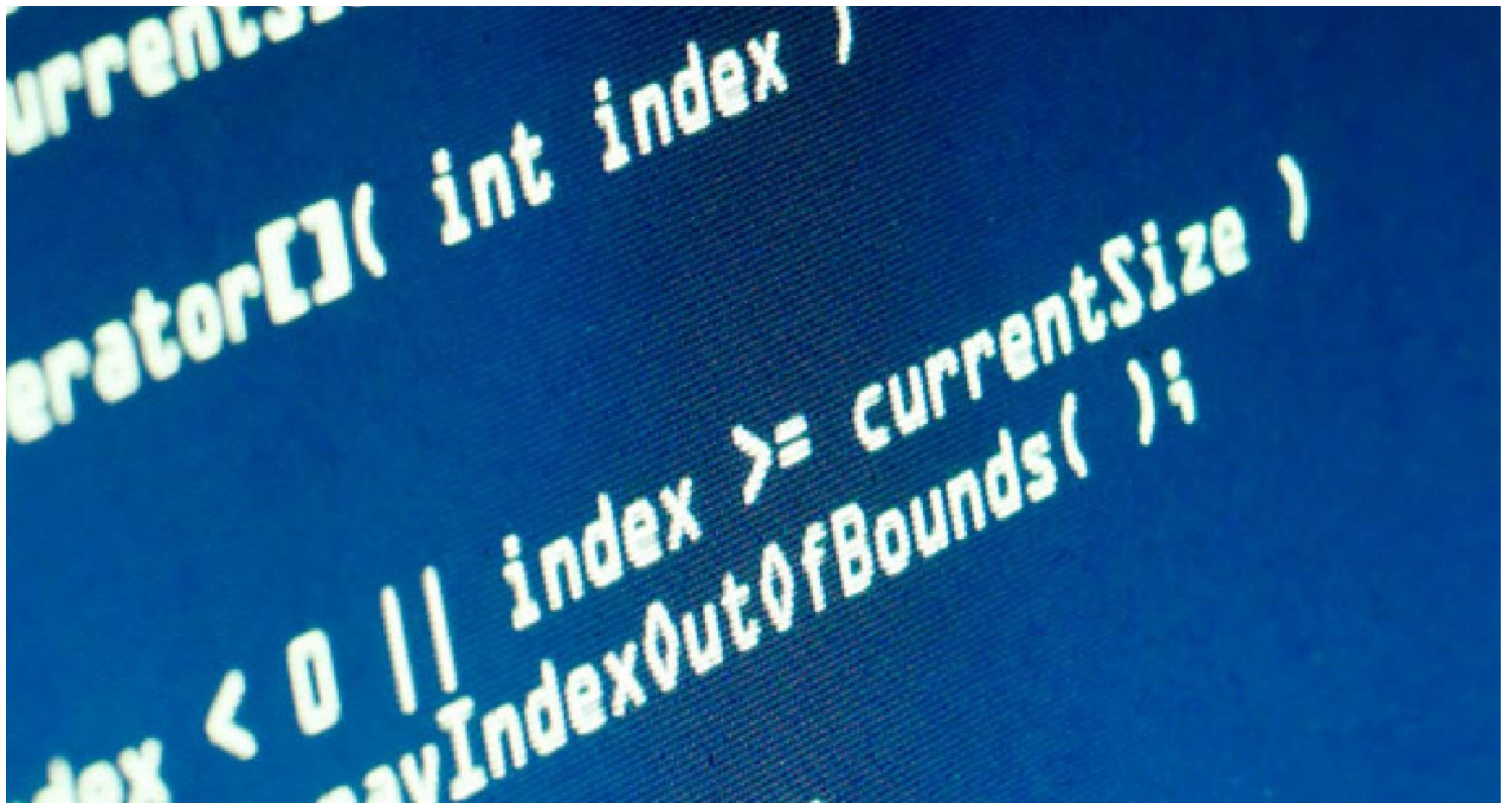
int size( ) const
{ return currentSize; }

Object & operator[]( int index )
{
    if( index < 0 || index >= currentSize )
        throw ArrayIndexOutOfBounds( );
    return objects[ index ];
}

Object & operator[]( int index ) const
{
    if( index < 0 || index >= currentSize )
        throw ArrayIndexOutOfBounds( );
    return objects[ index ];
}
#endif
#endif

```


Culori

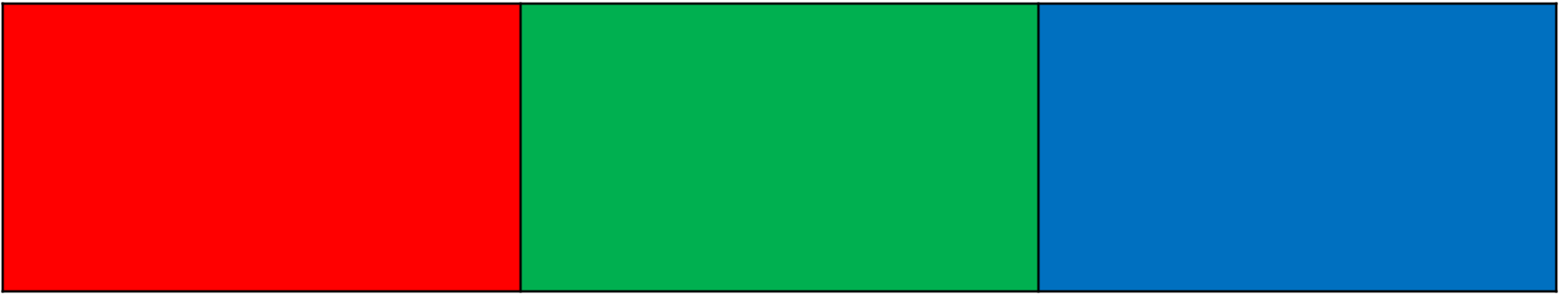


```
currentSize  
operator[]( int index )  
  
    if ( index < 0 || index >= currentSize )  
        throw IndexOutOfBounds( );
```

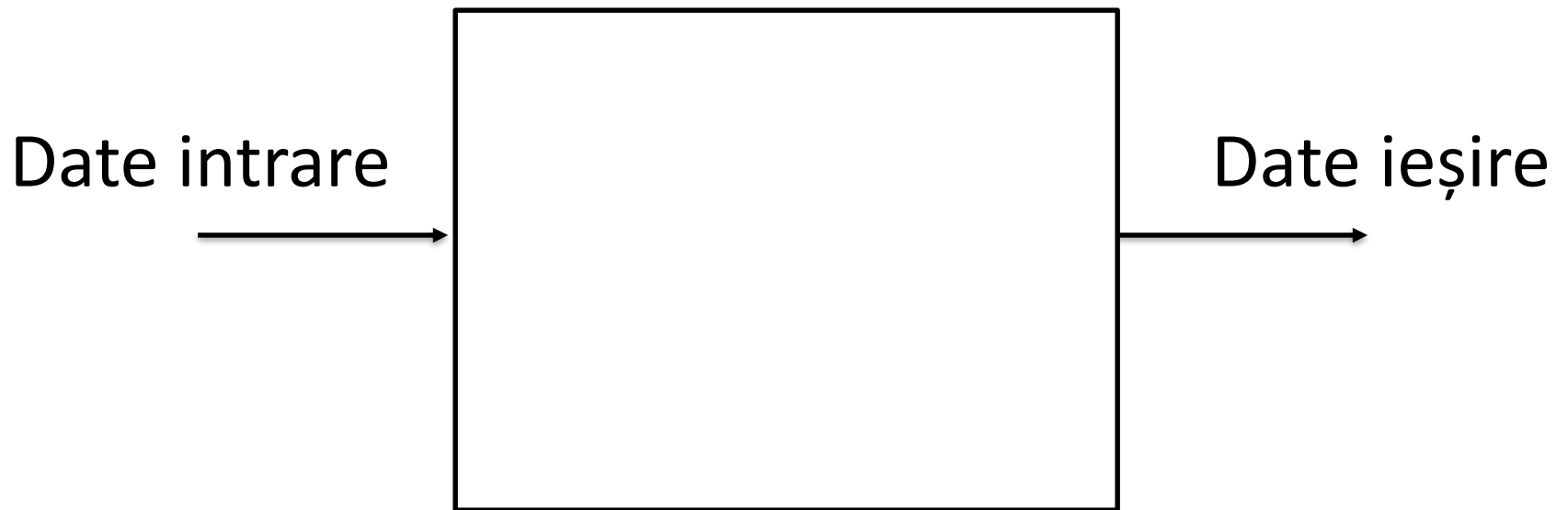
Culori

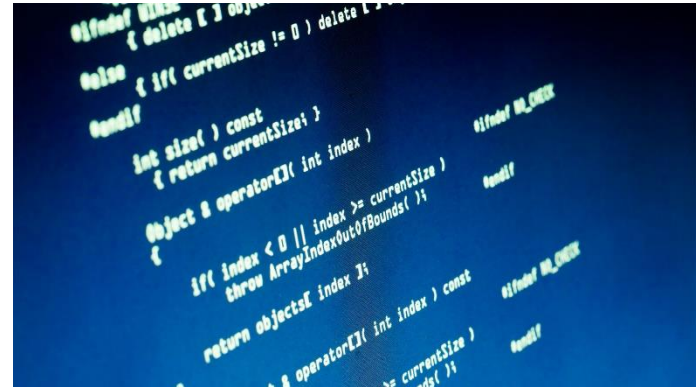


Culori



Structura unui algoritm





Ce facem la TP

Date în memoria calculatorului

Matrici

Matrici alocate dinamic

Exerciții

Matrici

Matricile au aplicații numeroase. Multe probleme din **teoria grafurilor, rețele neuronale, modelare sau grafică 3D** au la bază matrici.

La **Logică și Structuri Discrete - LSD** am văzut cum putem reprezenta relații sau grafuri cu matrici:

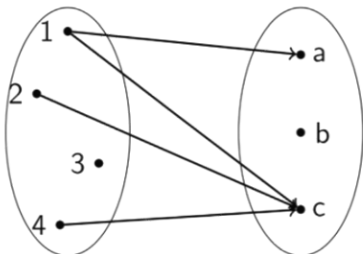
Reprezentarea unei relații

3. Ca **matrice booleană/binară**, dacă A, B finite,
– linii indexate după A , și coloanele după B

$m_{xy} = 1$ dacă $(x, y) \in R$,

$m_{xy} = 0$ dacă $(x, y) \notin R$

În practică putem folosi acest tip de reprezentare dacă A și B nu sunt foarte mari



	a	b	c
1	1	0	1
2	0	0	1
3	0	0	0
4	0	0	1

Reprezentarea grafurilor

Dacă identificăm nodurile prin numere (consecutive), putem reprezenta graful ca **matrice de adiacență** pătratică

$M[i,j] = 1$ dacă **există** muchie de la i la j

$M[i,j] = 0$ dacă **nu există** muchie de la i la j

sau $M[i,j]$ poate conține lungimea/costul muchiei (graf ponderat)

Matrici

În limbajul C se pot crea **vectori de vectori**.

Poartă denumirea de **vectori multidimensionali**.

Un **vector bidimensional** poartă denumirea de **matrice**.

```
int a[3][4];
```

Matrici

La fel ca la vectori, în general, elementele unei matrici sunt variabile, în schimb, în C, **matricea ca întreg nu poate fi tratată ca o variabilă**.

Nu putem, de exemplu, să atribuim unei matrici o altă matrice cu operatorul de atribuire, =.

Putem în schimb să **copiem element cu element** în bucle repetitive (for etc.).

Matrici

int a[3][4];

3 rânduri (de la 0 la 2),

4 coloane (de la 0 la 3),

12 elemente (3x4)

0 1 2 3

0

--	--	--	--

1

--	--	--	--

2

--	--	--	--

Inițializarea elementelor unei matrici

```
int a[2][3] = {{1, 3, 4}, {-3, 8, 12}};
```

	0	1	2
0	1	3	4
1	-3	8	12

Accesarea elementelor unei matrici

```
int a[2][3];
```

```
a[0][0] = 7;
```

```
a[0][1] = 2;
```

```
a[0][2] = -3;
```

```
a[1][0] = 3;
```

```
a[1][1] = 8;
```

```
a[1][2] = 12;
```

	0	1	2
0	7	2	-3
1	3	8	12

```
printf("%d", a[1][1]);
```

```
scanf("%d", &a[1][0]);
```

Accesarea elementelor unei matrici

```
#define N 2  
#define M 3
```

```
int a[N][M];
```

```
for(i=0; i<N;i++)  
    for(j=0; j<M; j++)  
    {  
        printf("a[%d][%d] =",  
            scanf("%d", &a[i][j]));  
    }
```

	0	1	2
0			
1			

Afişarea elementelor unei matrici

```
void afisare( int a[][M], int n, int m)
{
    for(int i=0; i<n;i++)
    {
        for(int j=0; j<m; j++)
            printf("%3d ",a[i][j]);
        printf("\n");
    }
}
```

Transmiterea ca parametrii

De regulă, în C, majoritatea variabilelor sunt **transmise ca valoare** la apelurile ca parametrii în funcții.

Matricile, la fel ca vectorii, nu respectă această regulă, ele **sunt transmise ca parametrii prin referință, nu prin valoare**.

Funcția care va prelucra ca parametru o matrice va primi **matricea originală**, nu o copie a acesteia.

Transmiterea ca parametrii

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

```
void set_int(int a) {  
    a=100;  
}
```

```
void set_matrix (int m[2][3]) {  
    m[0][0]=100;  
}
```

```
int main() {  
    int a=1;  
    int m[2][3]={{1,2,3},{4,5,6}};  
  
    printf("Initial: a= %d, m[0][0]=%d\n", a, m[0][0]);  
    set_int(a);  
    set_matrix(m);  
    printf("Final : a= %d, m[0][0]=%d", a, m[0][0]);  
    return 0;  
}
```

```
Initial: a= 1, m[0][0]=1  
Final   : a= 1, m[0][0]=100
```

Vectori și pointeri în contextul matricilor

O matrice este un **vector de vectori**.

Compilerul va amplasa în memorie elementele matricii **în ordinea liniilor, fără niciun spațiu între linii**:

a[0][0]	a[0][1]	...	a[0][N-1]	a[1][0]	a[1][1]	...	a[1][N-1]	...	a[M-2][0]	a[M-2][1]	...	a[M-2][N-1]	a[M-1][0]	a[M-1][1]	...	a[M-1][N-1]
linia 0				linia 1				...	linia M-2				linia M-1			

Adresa de memorie a elementului $a[i][j]$:

$$\&a[i][j] = \&a[0][0] + i * N + j$$

- $\&a[0][0]$ este adresa de început a matricii,
- $i * N$ este nr. de elem. conținut în liniile anterioare
- j este nr. de elem. până la indexul coloanei curente

Vectori și pointeri în contextul matricilor

a[0][0]	a[0][1]	...	a[0][N-1]	a[1][0]	a[1][1]	...	a[1][N-1]	...	a[M-2][0]	a[M-2][1]	...	a[M-2][N-1]	a[M-1][0]	a[M-1][1]	...	a[M-1][N-1]
linia 0				linia 1				...	linia M-2				linia M-1			

Matricea se poate parcurge cu un pointer **cu un singur for**, folosindu-ne de modul în care aceasta este reținută în memorie:

```
int *p;  
for(p=&a[0][0] ; p<&a[m][0] ; ++p)  
    *p=0;
```

Vectori și pointeri în contextul matricilor

În cele mai multe cazuri, când declarăm o matrice static, se va folosi doar o porțiune din matrice.

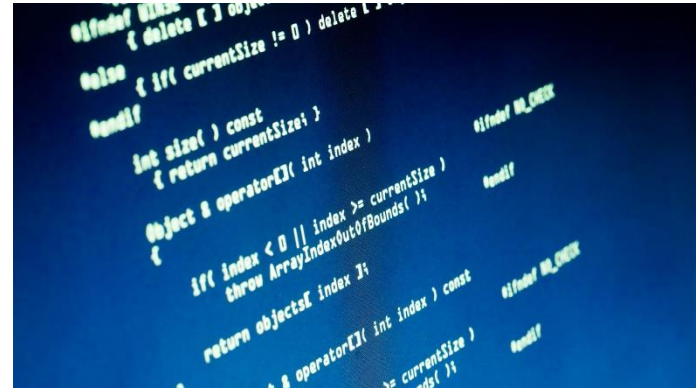
```
int main() {  
    int a[10][10], nr_linii, nr_coloane;
```

```
    scanf("%d",&nr_linii);  
    scanf("%d",&nr_coloane);  
    // daca se citesc nr_linii=2 si nr_coloane=3
```

```
    for(int i=0; i<nr_linii; i++)  
        for(int j=0; j<nr_coloane; j++)  
            a[i][j]=i+j;
```

```
    afisare(a, nr_linii, nr_coloane);  
    printf("\n");  
    afisare(a, 10, 10);  
    return 0;  
}
```

```
2  
3  
0 1 2  
1 2 3  
  
0 1 2 0 0 0 0 0 1 0  
1 2 3 0 0 0 0 0 0 0  
0 0 0 0 0 0 576 832 832 832  
896 1408 2432 2432 2432 2432 2432 2432 2432 2432  
2432 2432 2432 2432 2432 2432 2432 2432 2432 2432  
2432 2432 2432 2432 2432 2432 0 0 256 0  
0 64 512 1024 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 0  
13128008322208115775231 0 194 01666315319327641  
131280582122081-139072642432663131280574422081 0
```



Ce facem la TP

Date în memoria calculatorului

Matrici

Matrici alocate dinamic

Exerciții

Crearea unei matrici alocate dinamic

La fel ca vectorii, și matricile **se pot aloca dinamic**.

Avantajul folosirii matricilor alocate dinamic este că **se pot redimensiona** în funcție de necesități pe parcursul executării programului.

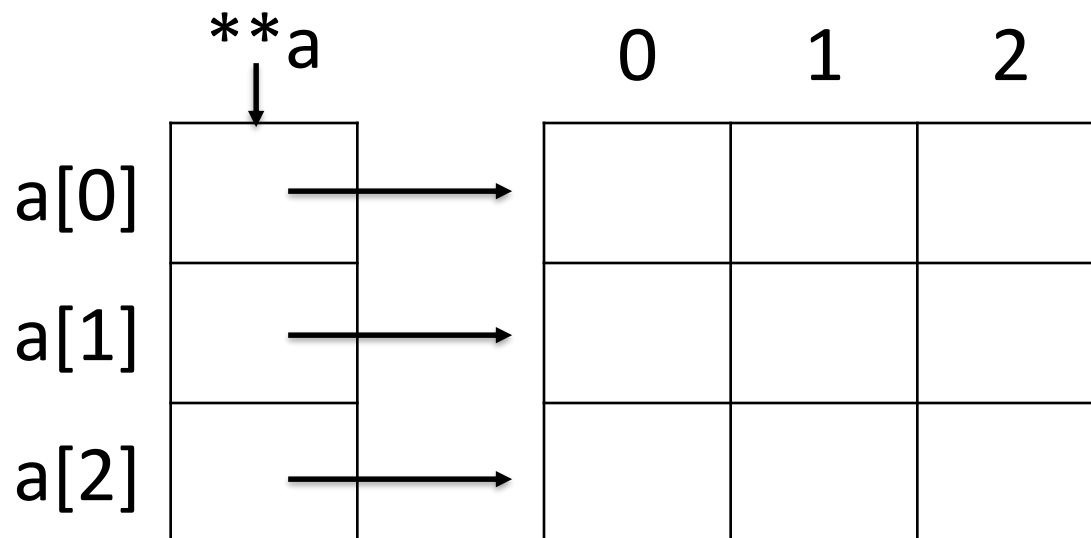
Putem să folosim un **vector de pointeri**, fiecare pointer pointând la câte o linie din matrice.

În acest caz **liniile matricii vor fi alocate separat**, ca blocuri de memorie independente.

Crearea unei matrici alocate dinamic

Matricea va fi definită ca `int **a`

`a[i]` este pointerul (`int*`) de la indexul `i` din `a`, adică adresa memoriei care conține acea linie



Crearea unei matrici alocate dinamic

```
int m,n,i,j;
```

```
int **a;
```

```
printf("m=");scanf("%d",&m);
```

```
printf("n=");scanf("%d",&n);
```

```
a=(int**)malloc(m*sizeof(int*));
```

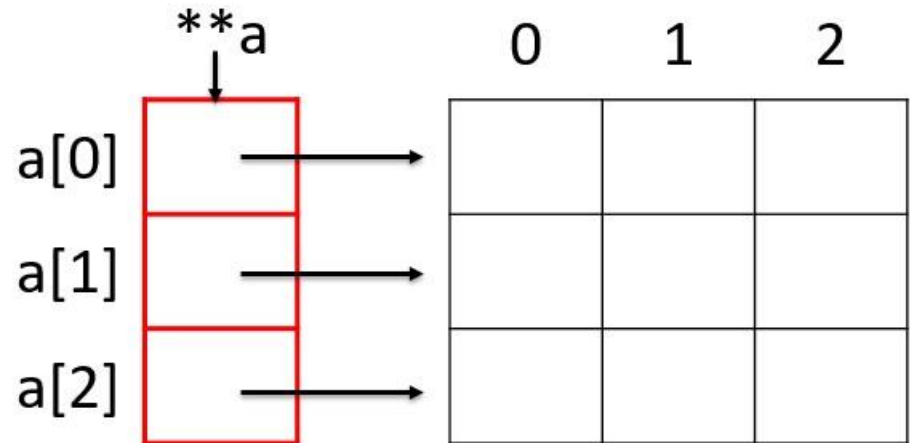
```
if(a==NULL)
```

```
{
```

```
    printf("memorie insuficienta\n");
```

```
    exit(EXIT_FAILURE);
```

```
}
```



Crearea unei matrici alocate dinamic

// alocare linii din matrice

```
for(i=0;i<m;i++)
```

```
{
```

```
a[i]=(int*)malloc(n*sizeof(int));
```

```
if(a[i]==NULL)
```

```
{
```

```
for(i--;i>=0;i--)
```

```
free(a[i]);
```

// elibereaza liniile alocate anterior

```
free(a);
```

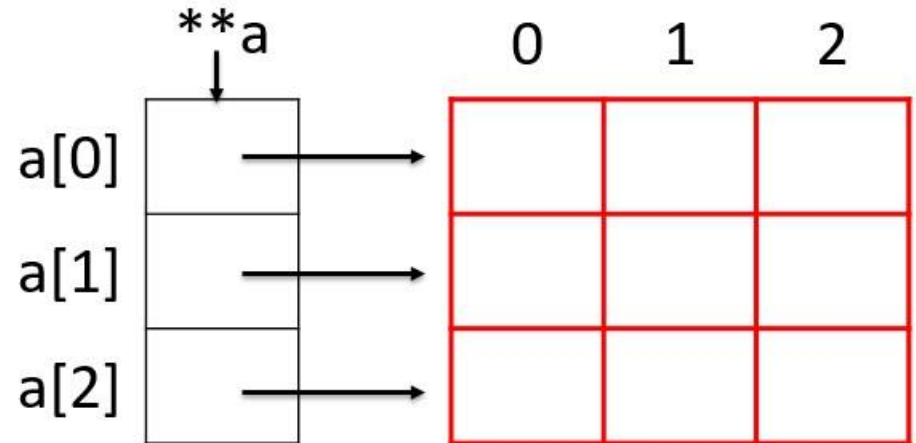
// elibereaza vectorul de pointeri

```
printf("memorie insuficienta\n");
```

```
exit(EXIT_FAILURE);
```

```
}
```

```
}
```



Eliberarea memoriei unei matrici alocate dinamic

Având în vedere că am alocat memorie dinamică, **e obligatoriu** ca la finalul programului **să o eliberăm**.

Înainte de a elibera cu **free(a)** e nevoie să eliberăm memoria ocupată de vectorii care reprezintă liniile matricii (**free(a[i])**):

```
for(i=0;i<m;i++)
```

```
    free(a[i]);
```

```
free(a);
```

```
// elibereaza fiecare linie
```

```
// elibereaza vectorul de linii
```


Alocarea dinamică a matricilor

Alocarea și eliberarea memoriei sunt mai complexe, pentru că fiecare linie se alocă separat.

În schimb, **accesarea elementelor matricii** se face exact ca în cazul în care ea ar fi fost declarată static (`int a[M][N]`), ceea ce simplifică toate operațiile cu matricea.



Ce facem la TP

Date în memoria calculatorului

Matrici

Matrici alocate dinamic

Exerciții

Exerciții

1. Se citesc de la tastatură m și n , fiecare mai mică decât 10. Să se creeze o matrice în care la fiecare poziție să fie maximul indecșilor acelei poziții și să se afișeze matricea.

Exerciții

2. Se citește un număr $n \leq 10$ de orașe și apoi pentru fiecare 2 orașe se citește distanța directă dintre ele. Dacă distanța este 0, înseamnă că între acele orașe nu există drum direct.

Să se afișeze perechea de orașe cele mai apropiate între ele în mod direct.

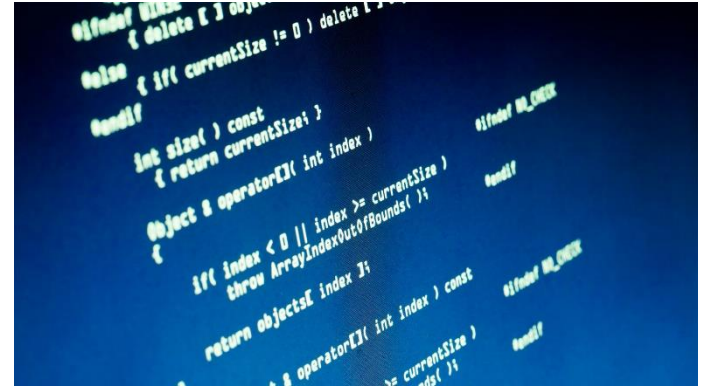
Exerciții

3. Se citește de la tastatură o matrice cu $n \leq 20$ linii și coloane.

Să se memoreze într-un vector suma tuturor elementelor de pe fiecare linie și într-un alt vector, suma tuturor elementelor de pe fiecare coloană.

Să se afișeze cei doi vectori.

Se vor utiliza pointeri atât pentru vectori cât și pentru matrici.



Vă mulțumesc!