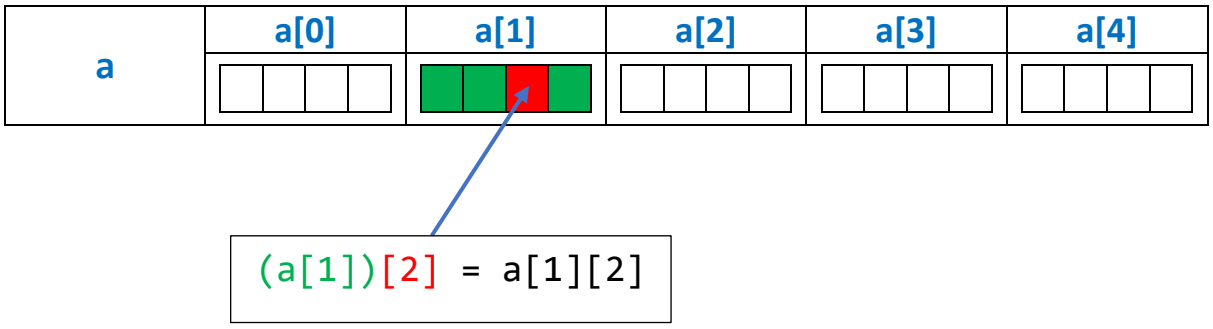
CURS 04 – PP
ALOCAREA DINAMICĂ A TABLOURILOR BIDIMENSIONALE

Tablou bidimensional = un tablou unidimensional ale cărui elemente sunt tablouri unidimensionale (liniile sale)

Exemplu:
```
int a[5][4];
```

`a[i]` = un tablou unidimensional reprezentând linia i a tabloului bidimensional



Observație: Elementele unui tablou bidimensional ocupă o zonă continuă de memorie!!!

Exemplu:
```
int a[5][4];
```

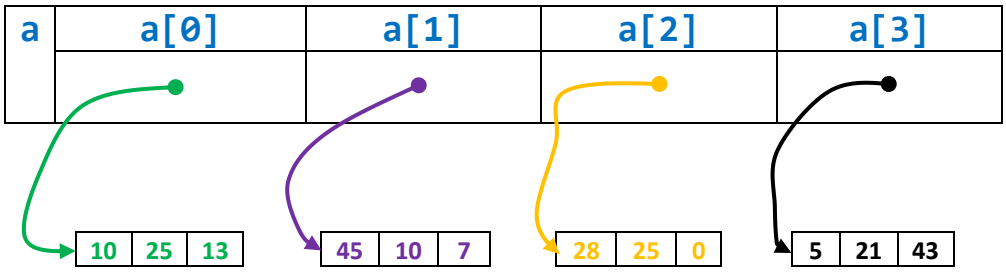
$m = 4$ linii și $n = 3$ coloane

$$a = \begin{pmatrix} 10 & 25 & 13 \\ 45 & 10 & 7 \\ 28 & 25 & 0 \\ 5 & 21 & 43 \end{pmatrix}$$

a	a[0]				a[1]				a[2]				a[3]				a[4]			
	10	25	13	?	45	10	7	?	28	25	0	?	5	21	43	?	?	?	?	?

? = valoare reziduală

ALOCAREA DINAMICĂ A UNUI "TABLOU BIDIMENSIONAL"



m = numărul de linii (4)
n = numărul de coloane (3)

a[i] = adresa de început a unei zone de memorie din HEAP unde a fost alocată linia i tabloului bidimensional

a = adresa de început a unui tablou unidimensional de adrese = adresa unei adrese = pointer dublu

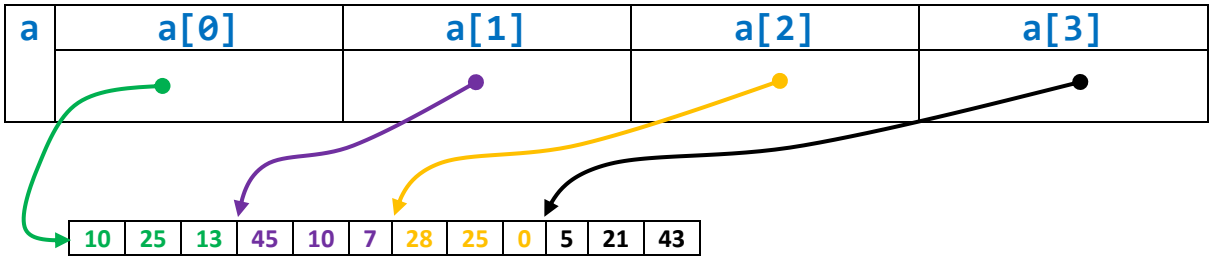
Alocarea dinamică necompactă a unui "tablou bidimensional" cu *m* linii și *n* coloane

```
//tabloul bidimensional este, de fapt, un pointer dublu
int **a;

//alocăm tabloul unidimensional "a" cu m elemente
//de tip int*
a = (int**)malloc(m * sizeof(int*));

//alocăm fiecare linie, adică un tablou unidimensional cu n
elemente de tip int
for(i = 0; i < m; i++)
     a[i] = (int*)malloc(n * sizeof(int));
```

Eliberarea zonei de memorie pentru un "tablou bidimensional" alocat necompact

```
//eliberăm zona de memorie alocată fiecărei linii
for(i = 0; i < m; i++)
     free(a[i]);

//eliberăm zona de memorie alocată tabloului bidimensional
free(a);
```

Alocarea dinamică compactă a unui "tablou bidimensional" cu *m* linii și *n* coloane




```
//tabloul bidimensional este, de fapt, un pointer dublu
int **a;

//alocăm tabloul unidimensional "a" cu m elemente
//de tip int*
a = (int**)malloc(m * sizeof(int*));

//alocăm dinamic spațiu de memorie pentru toate cele m*n
//elemente ale tabloului la adresa a[0]
a[0] = (int*)malloc(m * n * sizeof(int));

//în elementele a[1],...,a[m-1] memorez adresele din tabloul
//a[0] la care încep restul liniilor
for(i = 1; i < m; i++)
     a[i] = a[0] + n*i;
```

Eliberarea zonei de memorie pentru un "tablou bidimensional" alocat compact

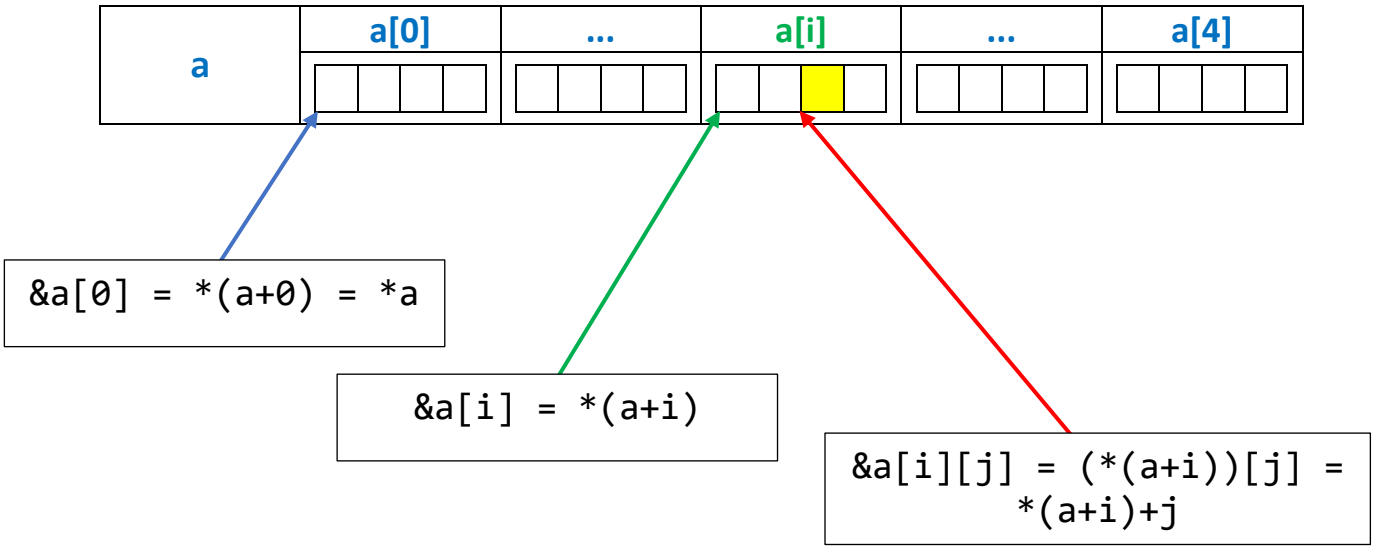
```
//eliberăm zona de memorie alocată tuturor elementelor
free(a[0]);

//eliberăm zona de memorie alocată tabloului bidimensional
free(a);
```

LEGĂTURA DINTRE TABLOURILE BIDIMENSIONALE ALOCATE STATIC ȘI POINTERI

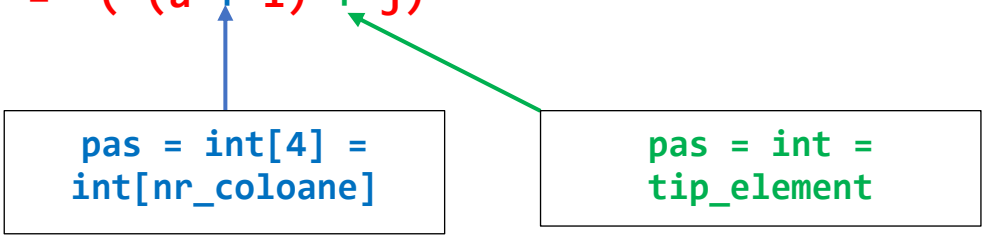
Numele unui tablou este adresa primului său element:
`a == &a[0] == &a[0][0]`

Exemplu:
```
int a[5][4];
```

`a[i]` = un tablou unidimensional reprezentând linia i a tabloului bidimensional



`a[i][j] = *(*(a + i) + j)`



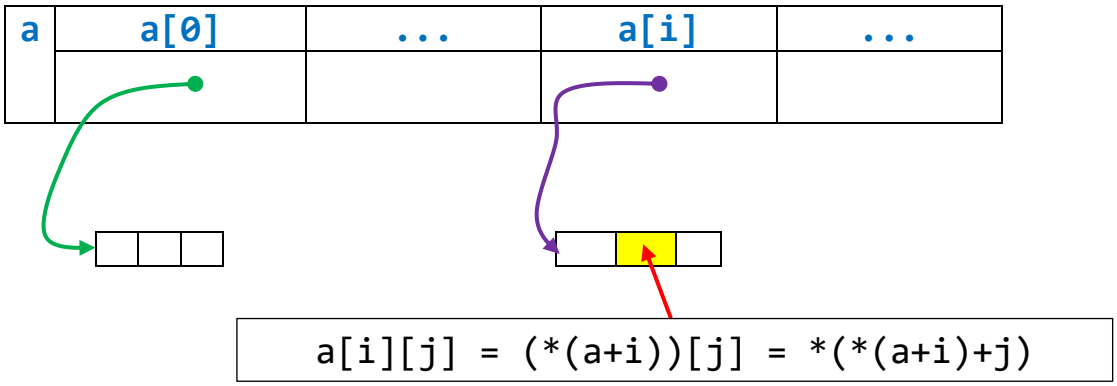
Un tablou bidimensional alocat static este echivalent din punct de vedere al accesării elementelor sale cu un pointer la un tablou cu numărul de elemente egal cu numărul de coloane, ci nu cu un pointer dublu!!!

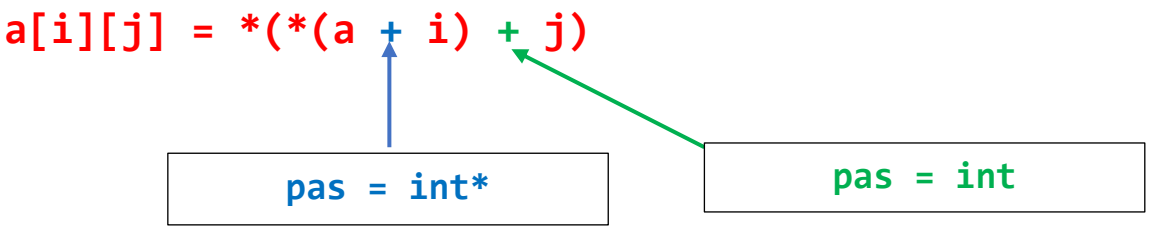
EXEMPLU:

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

//nu ma intereseaza numarul de linii,
//ci doar pasul de salt de la o linie la alta,
//adica numarul de coloane - CORECT!!!
void afisare_1(int t[][7], int m, int n)
{
    int i, j;

    printf("Matricea:\n");
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < n; j++)
            printf("%3d ", t[i][j]);
        printf("\n");
    }
    printf("\n");
}

//consider matricea un pointer dublu - INCORECT!!!
void afisare_2(int **t, int m, int n)
{
    int i, j;

    printf("Matricea:\n");
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < n; j++)
            printf("%3d ", t[i][j]);
        printf("\n");
    }
    printf("\n");
}

//consider matricea un pointer la un tablou
//cu 7 elemente de tip int - CORECT!!!
//int* t[7] = un tablou cu 7 elemente de tip int*
//int (*t)[7] = un pointer la un tablou cu 7 elemente de tip int
void afisare_3(int (*t)[7], int m, int n)
{
    int i, j;

    printf("Matricea:\n");
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < n; j++)
```

```
            printf("%3d ", t[i][j]);
        printf("\n");
    }
    printf("\n");
}

int main()
{
    int a[5][7] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};

    //matricea a:
    //1  2  3  4 0 0 0
    //5  6  7  8 0 0 0
    //9 10 11 12 0 0 0
    //0  0  0  0 0 0 0
    //0  0  0  0 0 0 0

    int m = 3, n = 4;

    //afisare_1(a, m, n);
    //afisare_2(a, m, n);
    afisare_3(a, m, n);

    return 0;
}
```

ACCESAREA ELEMENTELOR UNUI "TABLOU BIDIMENSIONAL" ALOCAT DINAMIC (COMPACT SAU NECOMPACT)





EXEMPLU:

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

//nu ma intereseaza numarul de linii,
//ci doar pasul de salt de la o linie la alta,
//adica numarul de coloane - INCORECT!!!
//pasul este pointer la int[3] in loc de int*
void afisare_1(int t[][3], int m, int n)
{
    int i, j;

    printf("Matricea:\n");
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < n; j++)
            printf("%3d ", t[i][j]);
        printf("\n");
    }
    printf("\n");
}

//consider matricea un pointer dublu - CORECT!!!
void afisare_2(int **t, int m, int n)
{
    int i, j;

    printf("Matricea:\n");
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < n; j++)
            printf("%3d ", t[i][j]);
        printf("\n");
    }
    printf("\n");
}

//consider matricea un pointer la un tablou
//cu 3 elemente de tip int - INCORECT!!!
//pasul este pointer la int[3] in loc de int*
void afisare_3(int (*t)[3], int m, int n)
{
    int i, j;

    printf("Matricea:\n");
    for(i = 0; i < m; i++)
    {
        for(j = 0; j < n; j++)
```

```
            printf("%3d ", t[i][j]);
        printf("\n");
    }
    printf("\n");
}

int main()
{
    int **a, i, j, m = 5, n = 3;

    //alocare necompacta
    //a = (int**)malloc(m * sizeof(int*));
    //for(i = 0; i < m; i++)
    //    a[i] = (int*)malloc(n * sizeof(int));

    //alocare compacta
    a = (int**)malloc(m * sizeof(int*));
    a[0] = (int*)malloc(m * n * sizeof(int));
    for(i = 1; i < m; i++)
        a[i] = a[0] + n*i;

    for(i = 0; i < m; i++)
        for(j = 0; j < n; j++)
            a[i][j] = i*n + j + 1;

    //tabloul bidimensional a:
    // 1  2  3
    // 4  5  6
    // 7  8  9
    //10 11 12
    //13 14 15

    //afisare_1(a, m, n);
    afisare_2(a, m, n);
    //afisare_3(a, m, n);

    //alocare necompacta
    //for(i = 0; i < m; i++)
    //    free(a[i]);
    //free(a);

    //alocare compacta
    free(a[0]);
    free(a);

    return 0;
}
```

EXPRESII ECHIVALENTE PENTRU ACCESAREA ELEMENTELOR UNUI TABLOU BIDIMENSIONAL (INDIFERENT DE MODUL DE ALOCARE)

```
a[i][j] = *(*(a+i) + j)
```

- `a[i][j] = *(a[i] + j)`
- `a[i][j] = (*(a + i))[j]`
- `a[i][j] = (*(a + i))[j] = (*(i + a))[j] = i[a][j]`
- `a[i][j] = *(*(a + i) + j) = *(j + *(a + i)) = j[*(a+i)]`
- `a[i][j] = j[*(a+i)] = j[a[i]]`
- `a[i][j] = j[*(a+i)] = j[*(i+a)] = j[i[a]]`

OBSERVAȚIE: În cazul unor "tablouri bidimensionale" alocate dinamic, NU mai este obligatoriu ca toate liniile să aibă același număr de elemente!!! Totuși, în acest caz, ar trebui să memorăm într-un alt tablou unidimensional numărul de elemente de pe fiecare linie!!!

EXEMPLU:
Scrieți o funcție care primește ca parametru un număr natural $n \geq 2$ și returnează un tablou bidimensional triunghiular alocat dinamic având următoarea formă:

$$
\begin{array}{llllll}
1 & & & & & \\
2 & 2 & & & & \\
3 & 3 & 3 & & & \\
\dots & \dots & \dots & \dots & & \\
n & n & n & \cdots & n & n
\end{array}
$$

Linia 0 are 1 element egal cu 1
Linia 1 are 2 elemente egale cu 2
Linia 2 are 3 elemente egale cu 3
.....
Linia i are i+1 elemente egale cu i+1
.....
Linia n-1 are n elemente egale cu n

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

void afisare(int **a, int n)
{
    int i, j;

    printf("\nMatricea:\n");
    for(i = 0; i < n; i++)
    {
        for(j = 0; j <= i; j++)
            printf("%3d ", a[i][j]);
        printf("\n");
    }
}

int** matrice(int n)
{
    int i, j, **a;

    a = (int**)malloc(n * sizeof(int*));
    for(i = 0; i < n; i++)
    {
        a[i] = (int*)malloc((i+1) * sizeof(int));
        for(j = 0; j <= i; j++)
            a[i][j] = i+1;
    }

    return a;
}

int main()
{
    int **a;
    int i, n;

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

    a = matrice(n);

    afisare(a, n);

    for(i = 0; i < n; i++)
        free(a[i]);
    free(a);

    return 0;
}
```