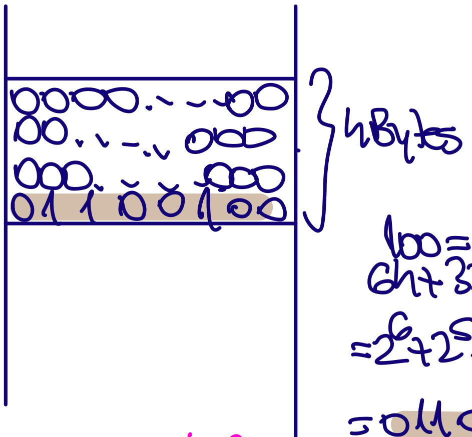
Curs 6 - Pointeri. Alocare dinamică

Pointer: un tip de dată care poate stoca adresele altor variabile / date. S-a introdus pointerul pentru a stoca adrese de memorie pe 32 biți (x86-32 biți), 64 de biți.

```
int x = 100;
int* px = &x;
```



$100 = 64 + 36 =$
$64 + 32 + 4$
$= 2^6 + 2^5 + 2^2$
$= 0110.0100$
76543210

Semnul / Operatorul * are două semnificații

1. Dacă se pune după tipul de dată int* px;
declerea unei variabile de tip pointer

```
int *px;
```

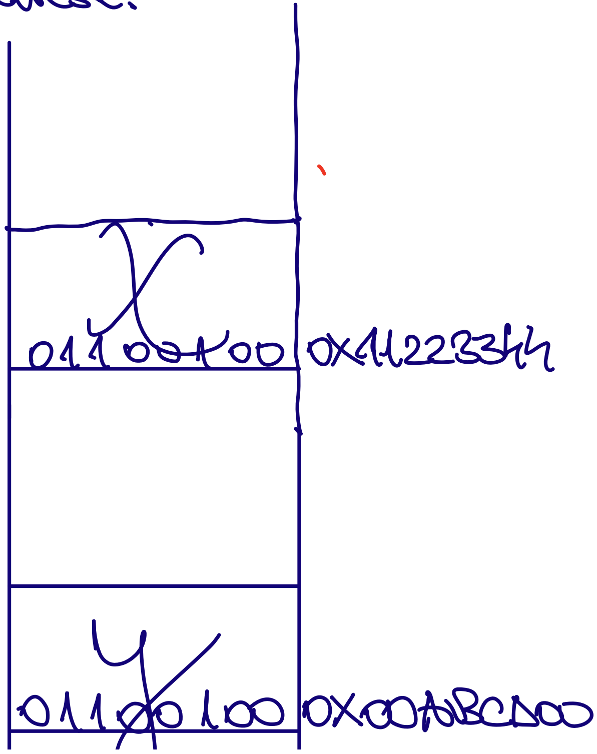
2. Înainte de o variabilă de tip pointer/adresă, dacă se trece operatorul *, înseamnă dereferențiere, se scoate valoarea stocată la acea adresă.

Inversul operatorului de dereferențiere este & pus în fața unei variabile. A nu se confunda cu bit cu bit & (și) cu && (și logic în condiții)

```
int x = 100;
int* adrx = &x;
```
$\underbrace{adrx}_{} = \underbrace{\&x}_{}$ 0X11223344 ↳ adresa lui x

~~int y = *(0X11223344);~~

```
int y = *adrx;
```



100

32 biți : Adresare pe 32 de biți grupuri de câte 4 → 8 cifre hexa

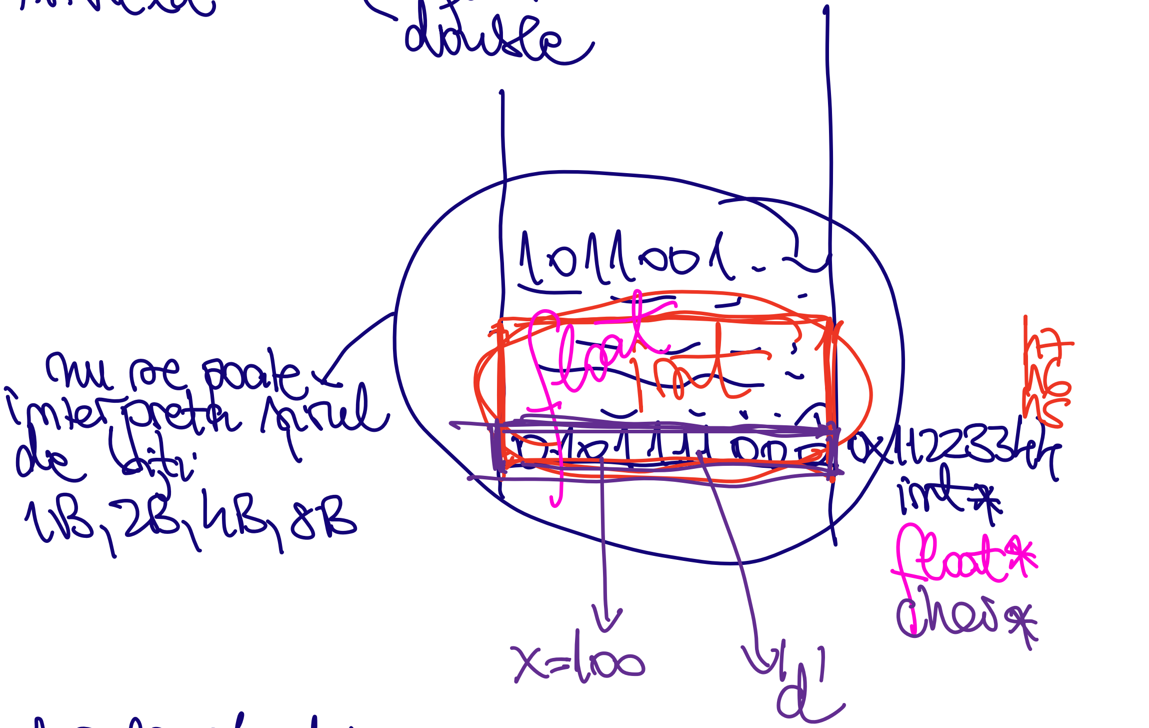
64 biți : — " — 64 ———— " ———— → 16 cifre hexa

$\text{Variabila} \xrightarrow{\&} \text{adresă}$ scanf("%d", &x);

$\text{Adresă} \xrightarrow{*} \text{valoarea stocată la acea adr.}$

De ce avem nevoie de tipul pointerului?

$\underbrace{0x11223344}_{\text{nr în hexa}}$, adresă ← se stochează char / int / float / double



Pointeri constanți

5a. Pointer la valori constante

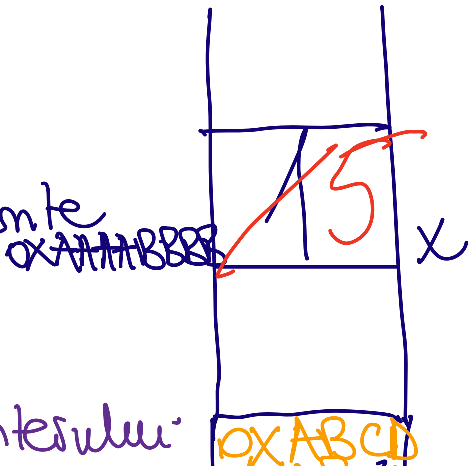
5b. Pointer constant

5c. Pointer constant la val. constante

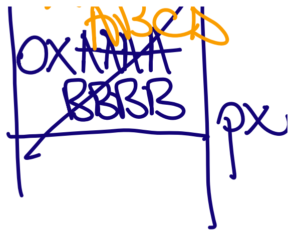
5a. ~~int x=1;~~ const int x=1;

const int* px=&x;

↳ înainte de tipul pointerului



*px=5; GRESITĂ valoarea nu se poate schimbe prin intermediul pointerului.



px=&y; CORECTĂ

5b. Pointer constanți

int x=1, y=2;

int* const px=&x; //prima atribuire de adr. rămâne fixă

↳ între tip și denumire de variabilă

*px=5; CORECT

px=&y; GREȘIT

5c. int x=1, y=2;

const int* const px=&x;

*px=5; GREȘIT

px=&y; GREȘIT

Legătura pointer tablou

int vector[100];

cout<<vector; //NU AFIȘEAZĂ TOATE EL. VECTORULUI

↳ adresa de bază a vectorului

adresa celui de al 0-lea element &v[0]

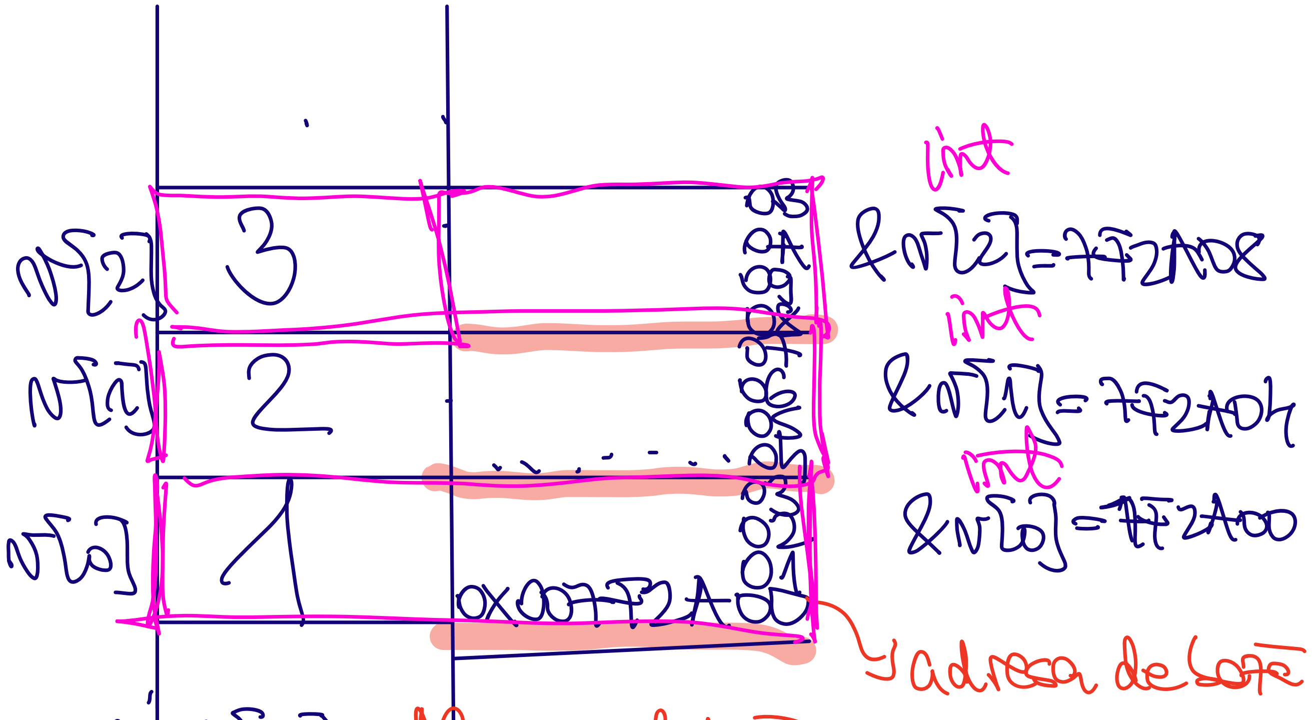
printf("%p", vector); //0X007FF2A00

Numele vectorului este adresa de bază.

v ⟺ &v[0] Adresa lui v/vectorului este adresa celui de al 0-lea el.

$$\boxed{(v+i) \Longleftrightarrow \&v[i]}$$

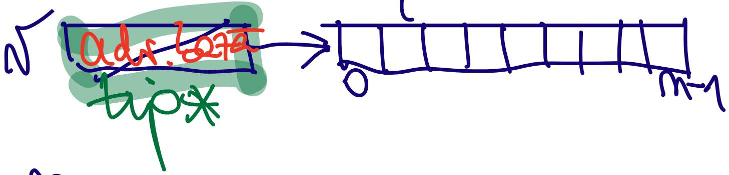
int v[3]={1,2,3};



int v[3]; Alocare statica
↳ valoare constanta

S-a introdus alocarea dinamica: malloc, calloc, realoc

Avantaj se poate aloca zona de memorie pt. vector în funcție de o variabila / val. variabilei



```
int n; citire n;
int* v;
v=(int*)malloc(n*sizeof(int));
```

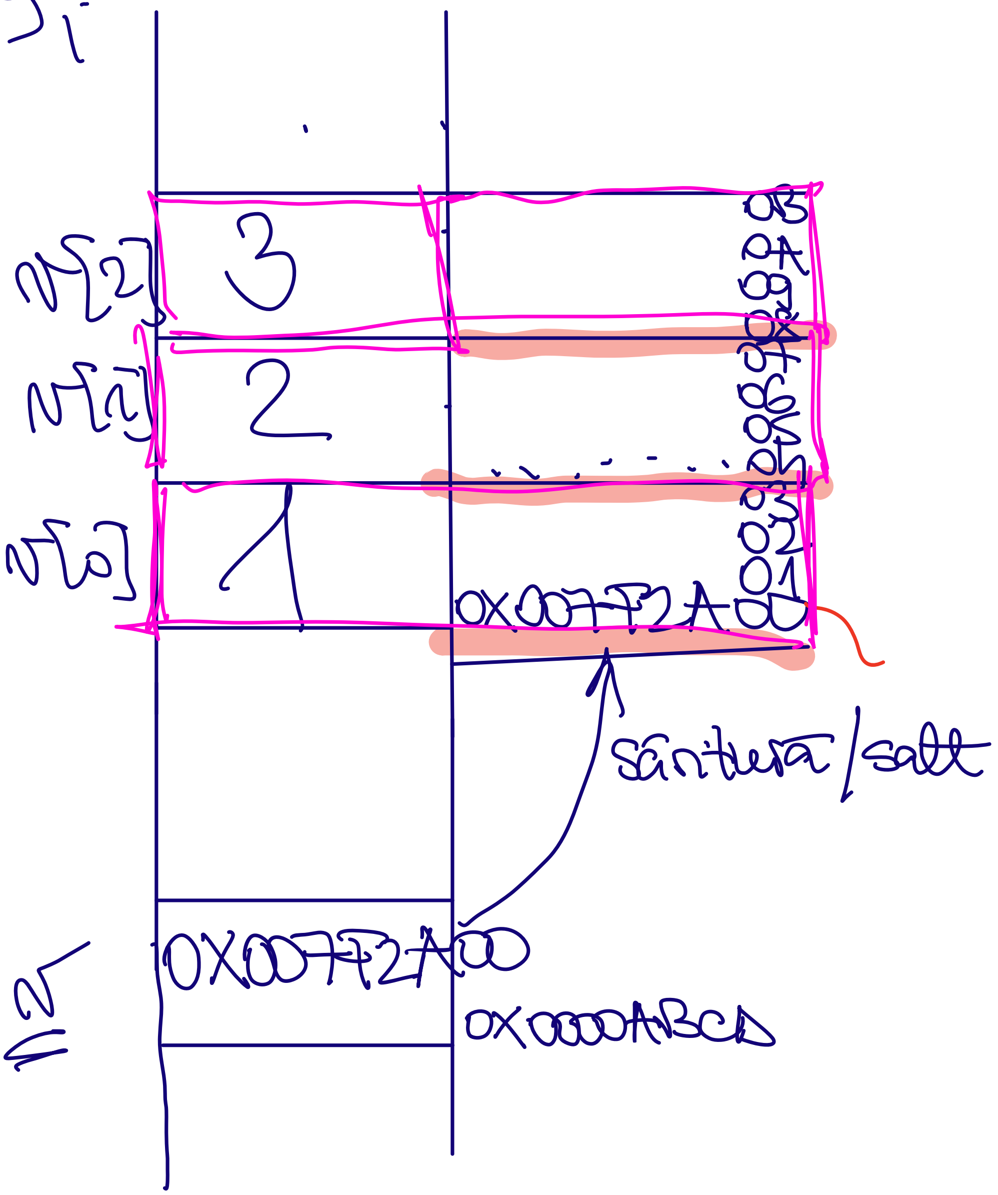
conversie de tip

malloc aloca memorie void*, zona de memorie generala

se converteste în zona de memorie pt.

int-uri

free(v);

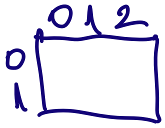


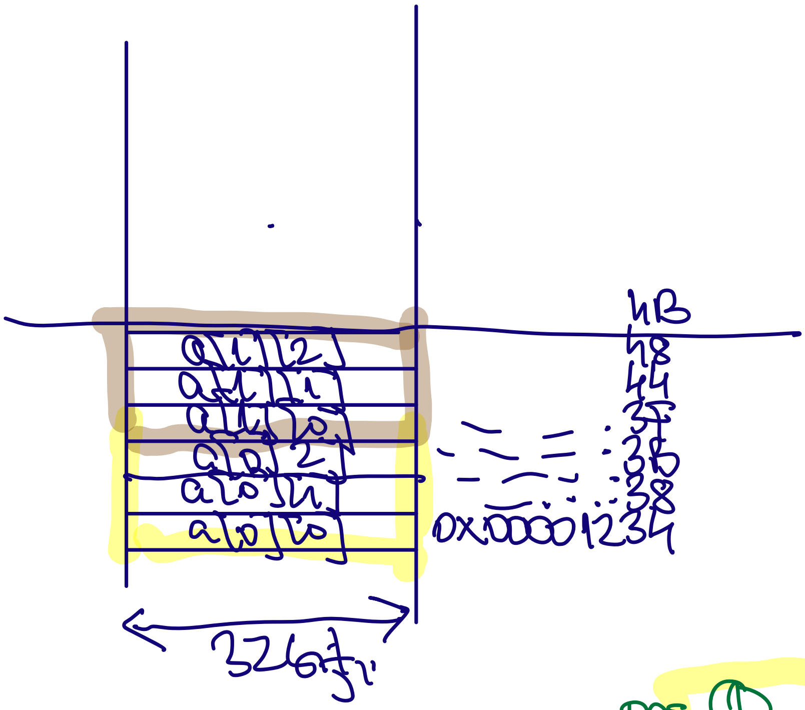
Matrice / Alocare dinamică

int a[100][100]; //alocare statica

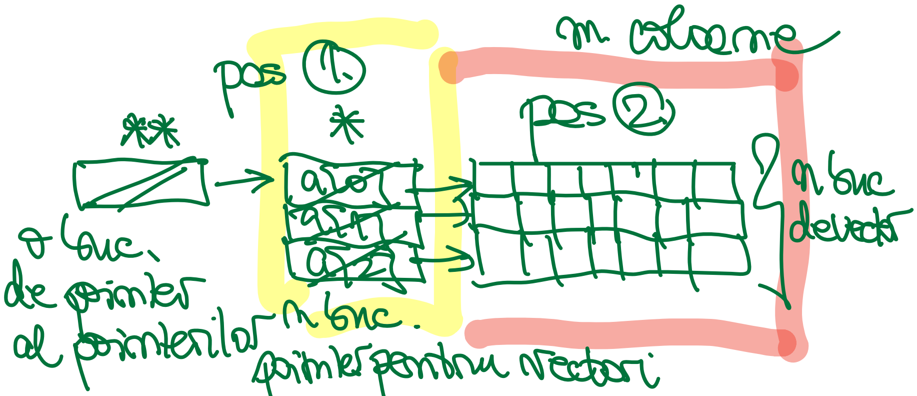
$\underbrace{\text{int a[2][3];}}$

nu se fixează nr. linii și coloane





```
int **a;
```


m coloane
pos ①
pos ②
**
*
a[0]
a[1]
a[2]
n buc de vect
o buc. de pointer al pointerilor
n buc. pointer pentru vectori

```
a= (int**) malloc(n* sizeof(int*)); // pos 1.
for(int i=0; i<n; ++i)
        a[i]= (int*) malloc(m* sizeof(int));    } pos 2.
```

Eliberare

```
for(int i=0; i<n; i++)
        free(a[i]);    } pos 2 eliberare
free(a); // pos 1.
```

$a[i][j] \Longleftrightarrow *(a[i]+j) \Longleftrightarrow *(*(a+i)+j)$

$v[i] \Longleftrightarrow *(v+i)$

&a[i][j] <=> ~~&*~~(*(a+i)+j) adresa

Curs 7 - Baze de numeratie
Operații la nivel de bit

Reprezentarea numerelor fără semn

$234_{(10)} = 11101010_{(2)}$ 8 biți: unsigned char

$234_{(10)} =$ 0000.0000.1110.1010 16 biți: unsigned short

MSB pentru semn, bitul de semn

+ pozitiv 0

- negativ 1

8 0 E A

1) Codul direct $-234_{(10)} =$ 1000.0000.1110.1010

2) Cod invers: În afară de bitul de semn se neagă fiecare bit

$-234_{(10)} =$ 1111.1111.0001.0101

```
 234 = 0000.0000.1110.1010 +
-234 = 1111.1111.0001.0101
-----------------------------
 0-  = 1111.1111.1111.1111 ) +1
 0+  = 0000.0000.0000.0000
-----------------------------
   [1] 0000.0000.0000.0000
   16 15                    0
```

3) Codul complement față de 2

Se obține din codul invers, la care se adună +1.

$-234_{(10)} =$ 1111.1111.0001.0101 + 1