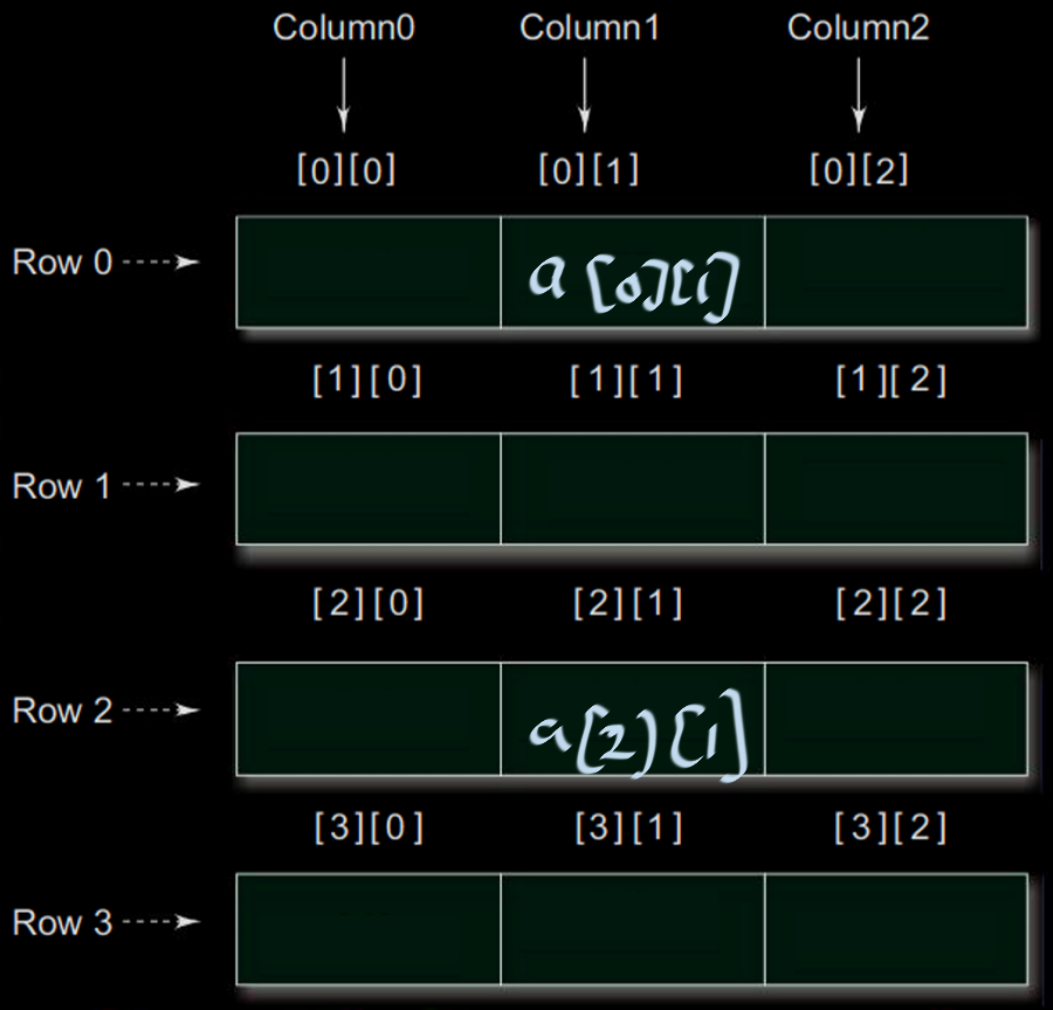
2 D arrays

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

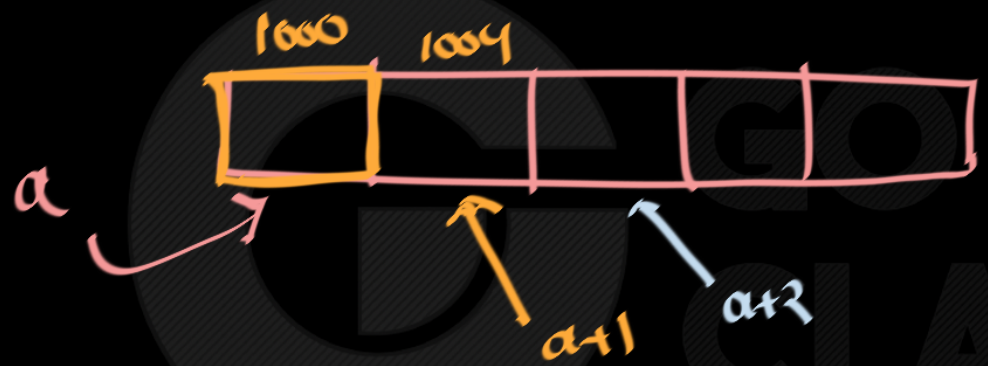
No. of rows

No. of columns



GO CLASSES

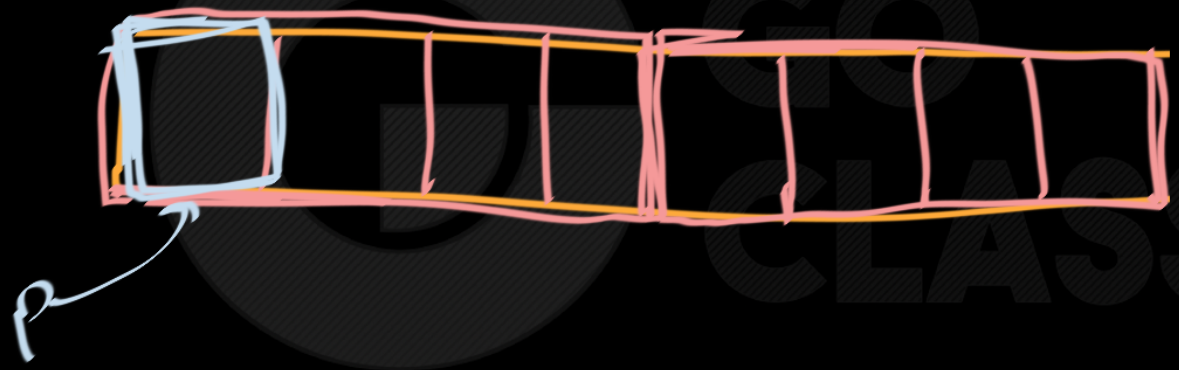
1 D array



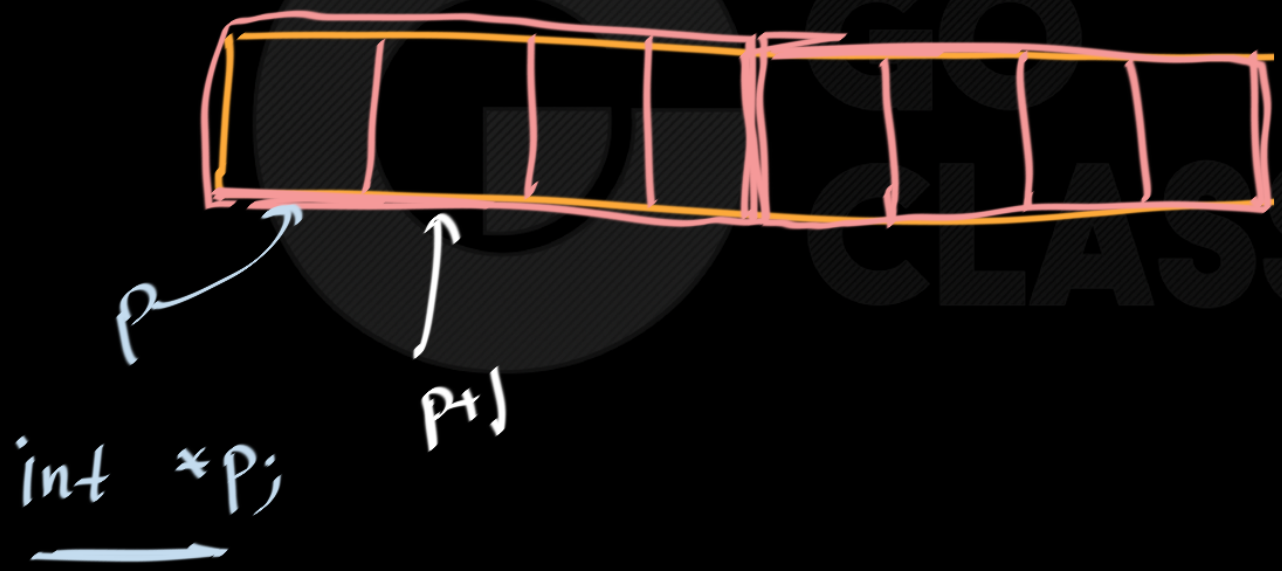
int a[5];

a ≡ 1000

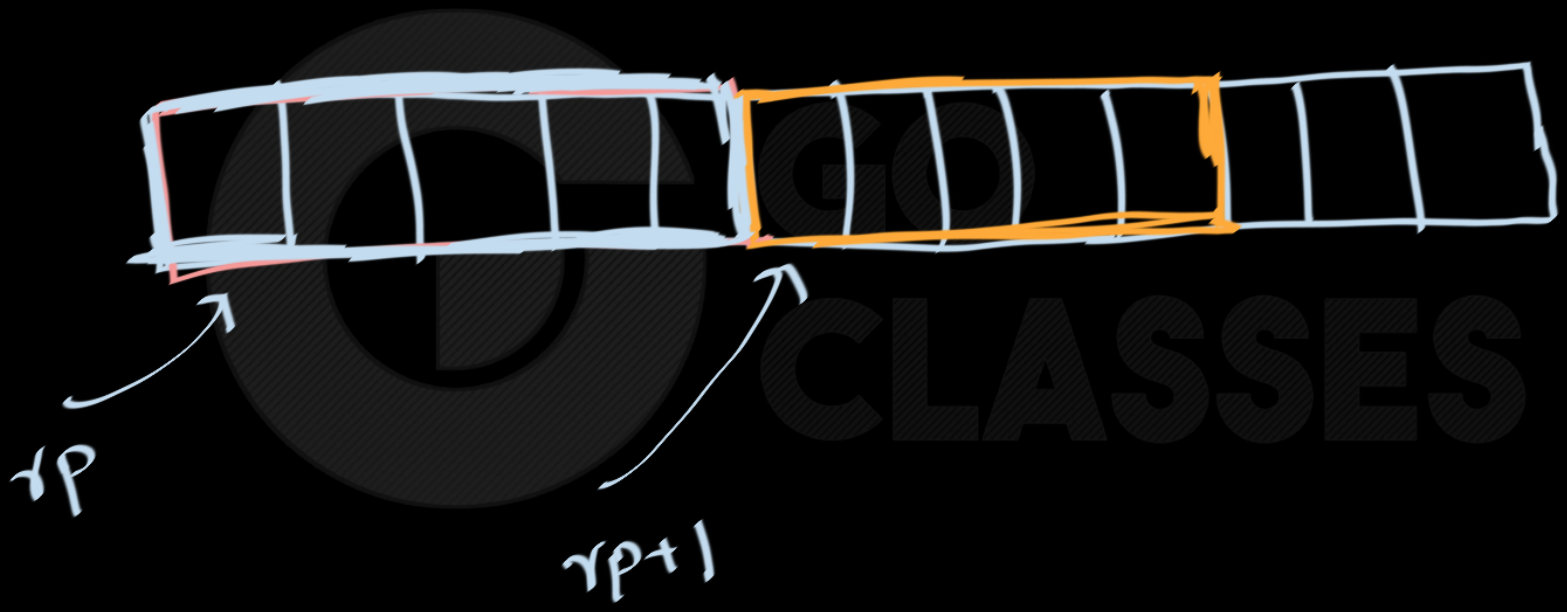
a+1 ≡ 1004



int *p;

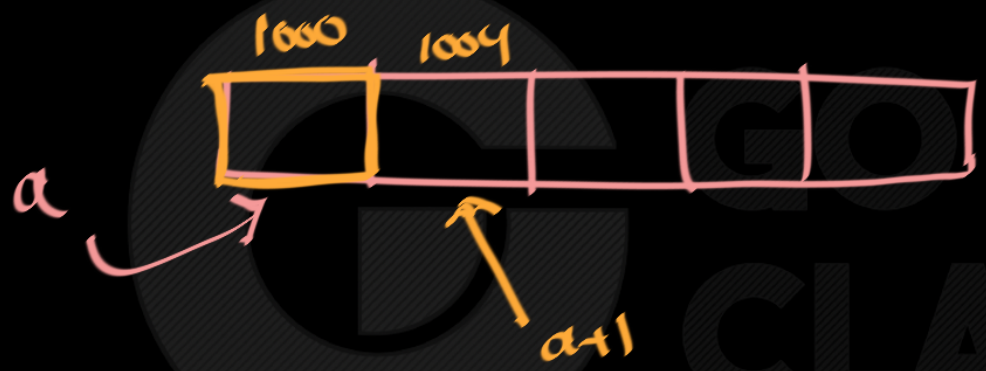

p
p+1
int *p;


```
int *rp;      Wrong

int (*rp)[5]; Right
```


rp
rp+1

GO CLASSES

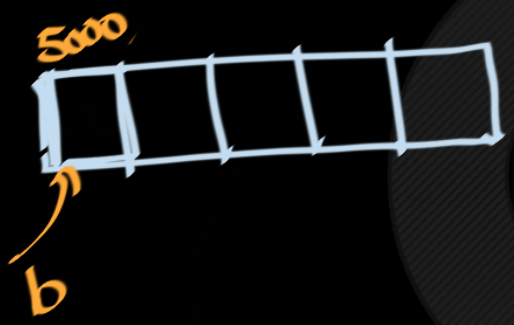
1D array



a ≡ 1000

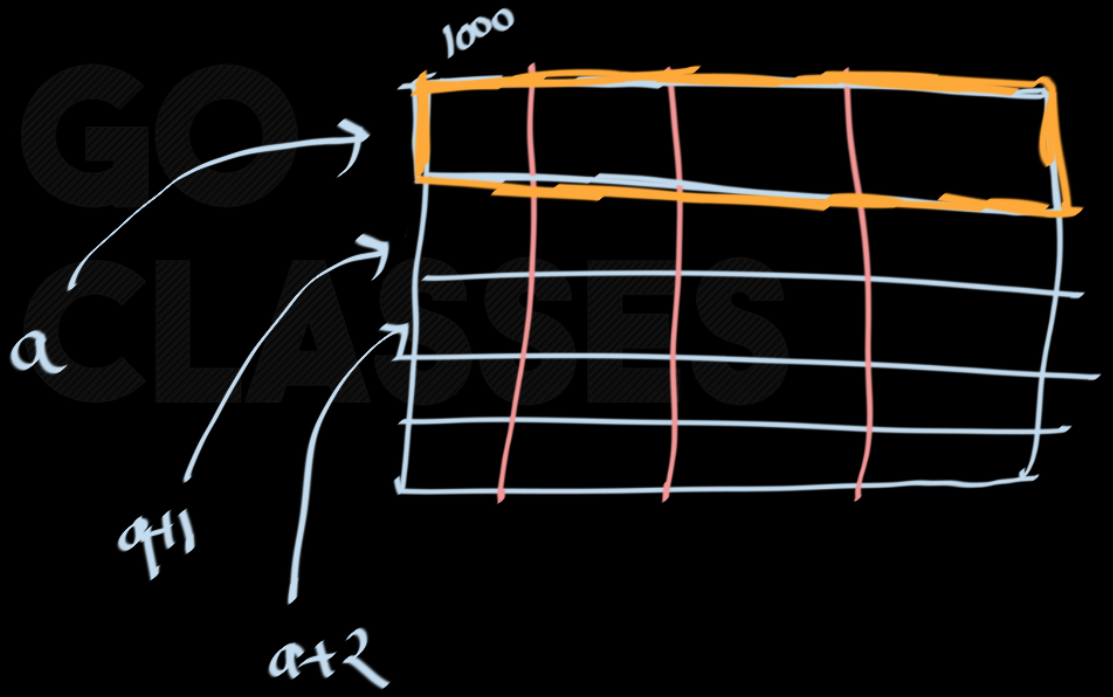
GO CLASSES

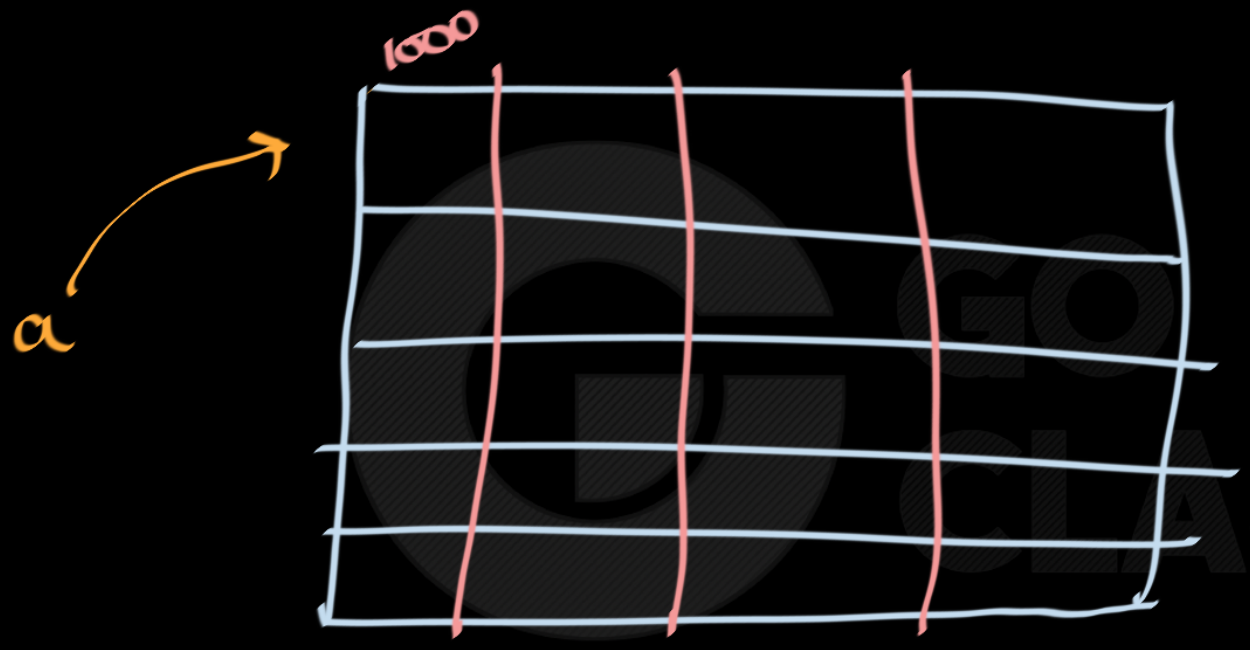
int b[5];



b ≡ 5000

int a[5][4];



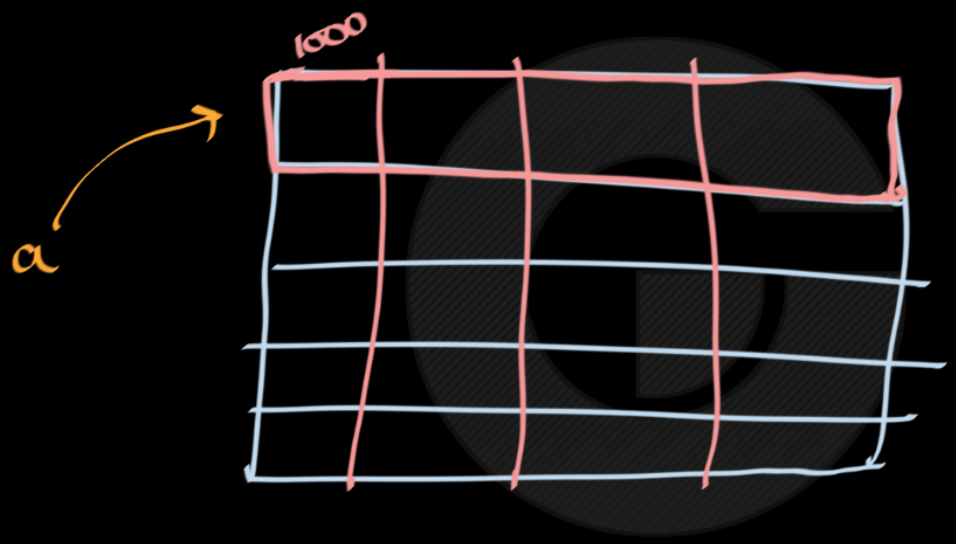


int a[5][4];

a ≡ 1000

a : Points to the very first row

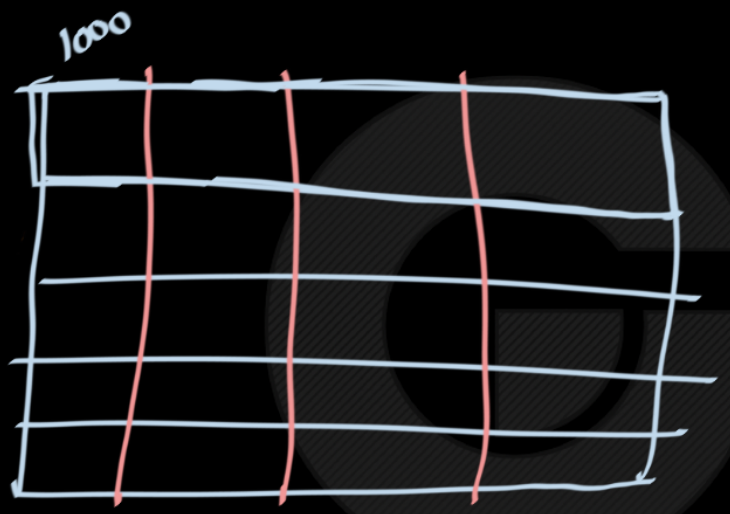
int a[5][4];



a : Points to the very first row

; is an address of first row

≡ 1000



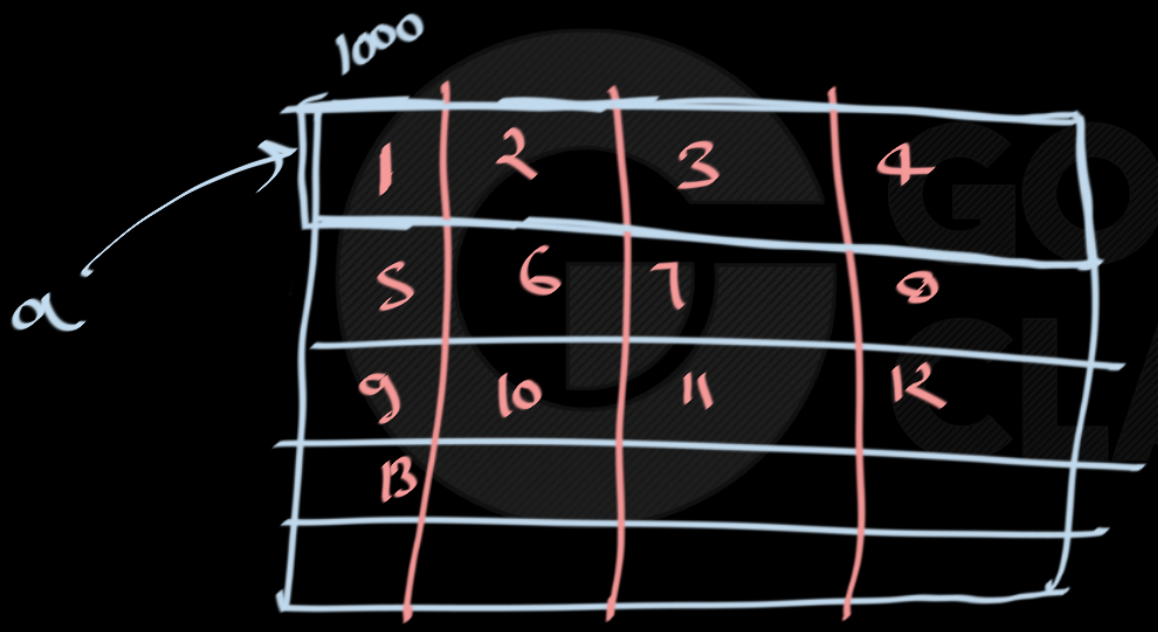
1000 is ⇒

(i) address of 1st integer

(ii) address of 1st row

(iii) address of 1st Byte

(iv) address of the entire 2D array



$a \equiv 1000$

$*a \equiv 1$? NO.

*a is pointer to integer.

int b[5]

b → address / pointer

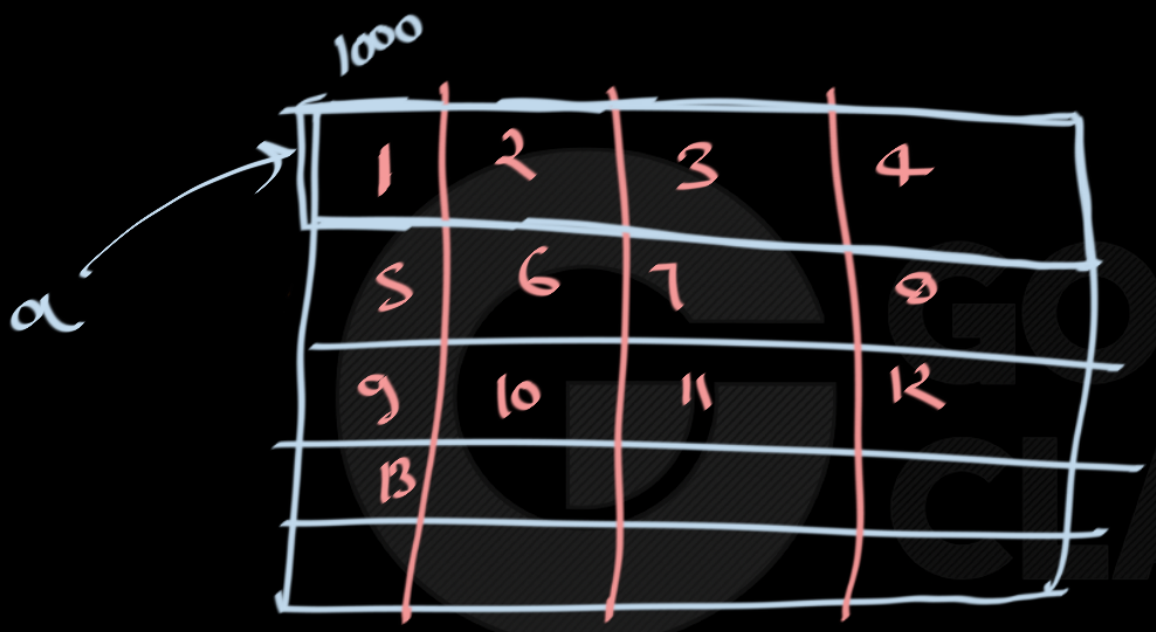
b[3] → integer

*b+1 ⇒ integer

int a[3][4]

[ ] [ ] } integer
[ ] *
* *

a[0][1] ✓ *a[1] ✓ *(*(a+1)+2) ✓

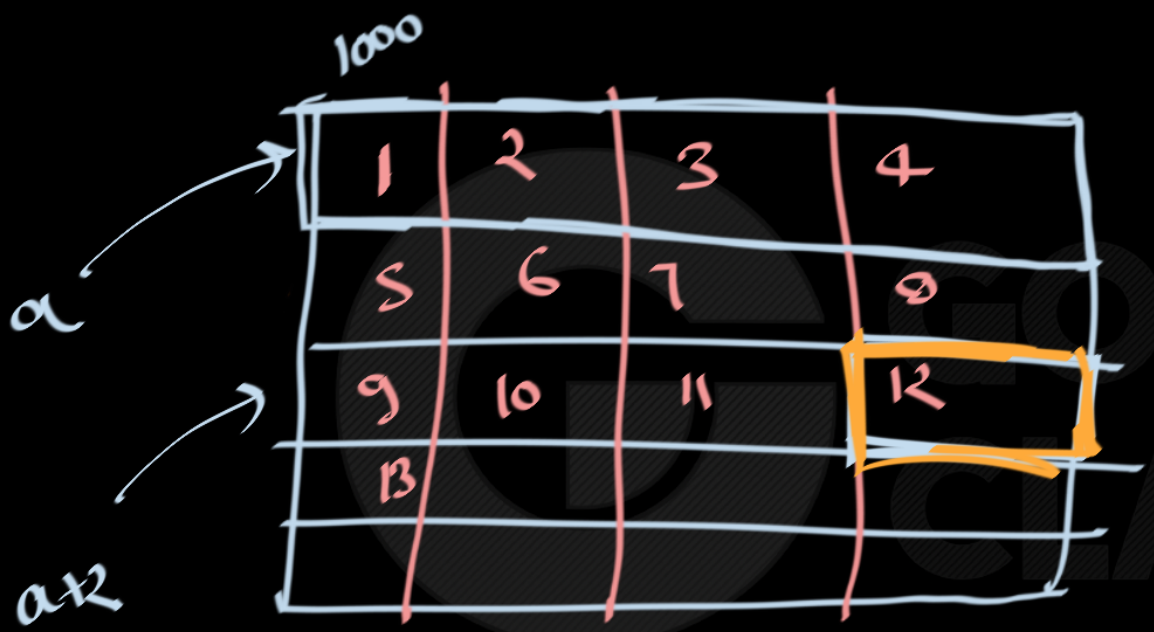


$a \equiv 1000$ ← add. of 1st row

$*a \equiv 1000$ ← add of 1st integer

$**a \equiv 1$

$\&a \equiv 1000$ ← add of entire array



$a[2][3] \equiv 12$

$*(*(a+2)+3)$

$*\left(*(a+2)+3\right) \equiv 12$

$*(a[2]+3) = a[2][3]$

GO CLASSES

in one D array

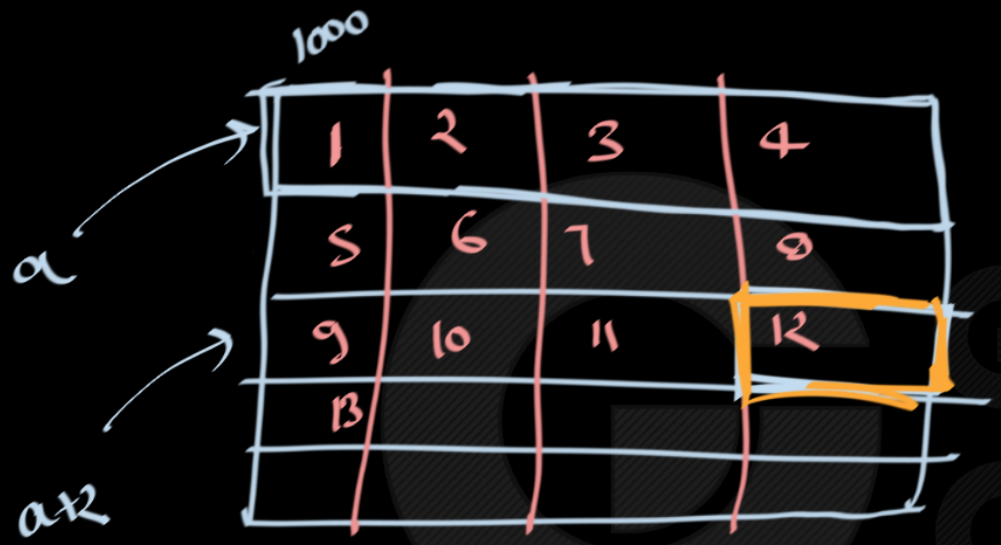
```
int b[5];
```

$b[i] \equiv *(b+i)$

in 2D array

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

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



$a[2][3]$

or

$\equiv R$

$*(*(a+2)+3)$

$*(*(a+1)+7)$

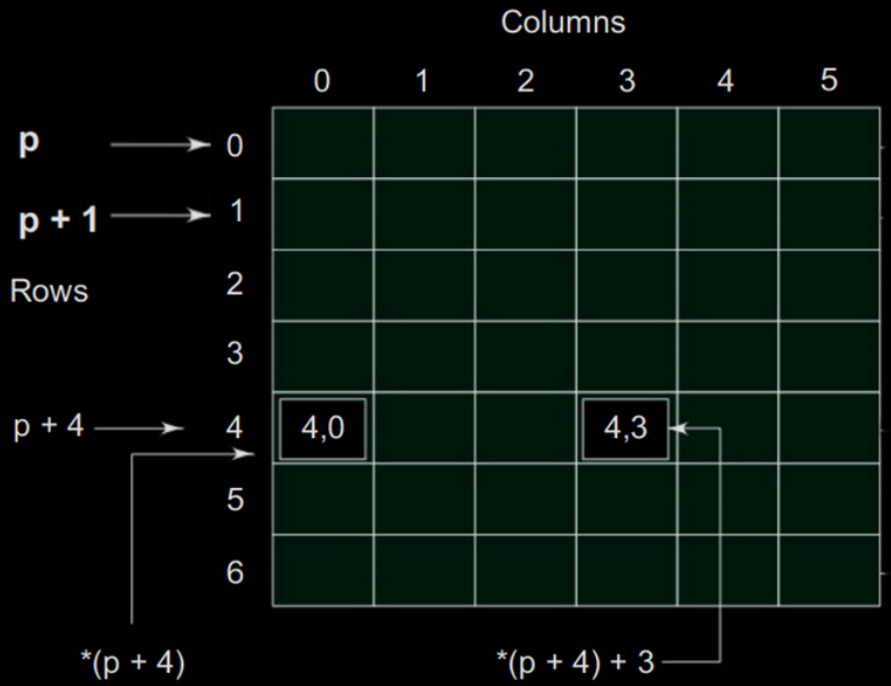
$a[1][7] \equiv R$

$*(*(a+3)-1)$

$a[3][-1] \equiv R$

$*(*a+11) \equiv R$

$a[0][11] \equiv R$



p	→	pointer to first row
p + i	→	pointer to ith row
*(p + i)	→	pointer to first element in the ith row
*(p + i) + j	→	pointer to jth element in the ith row
((p + i) + j)	→	value stored in the cell (i,j) (ith row and jth column)

Pointers to two-dimensional arrays

Question

MSQ

```
int x[3][4] = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
```

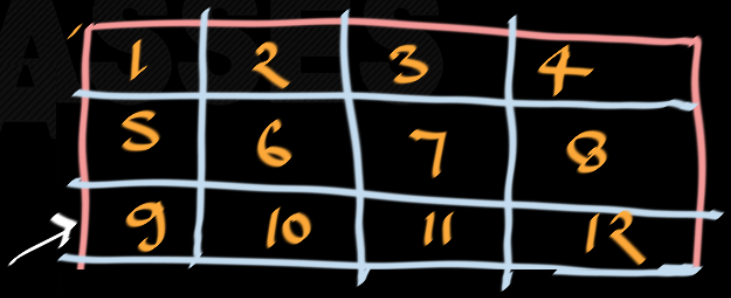
Which of the following print "7" ?

A. `printf("%d", x[1][2]);`

B. `printf("%d", *(*(x+1)+2));`

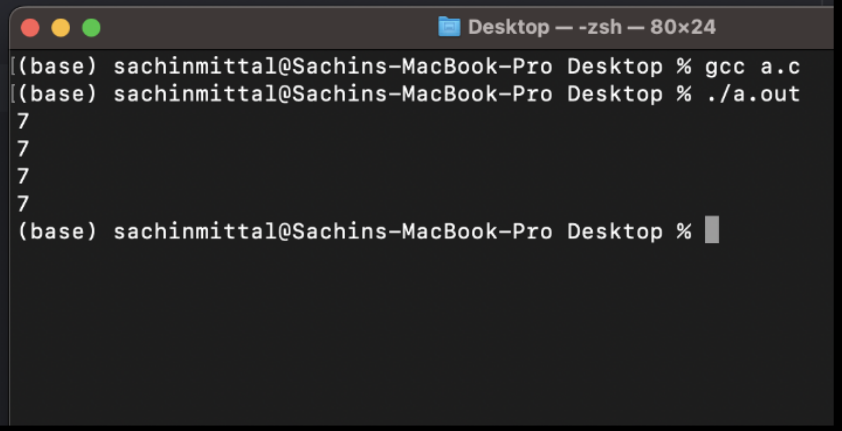
C. `printf("%d", *(*x+6));`

D. `printf("%d", *(*(x+2)-2));`



GO CLASSES

```c
#include<stdio.h>

int main(){


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

    //another way to initialise
    //int x[3][4] = {1 ,2 ,3 ,4 , 5 , 6 , 7 , 8 , 9 , 10 , 11,12}


    printf("%d\n", x[1][2]);
    printf("%d\n", *(*(x+1)+2));
    printf("%d\n", *(*x+6));
    printf("%d\n", *(*(x+2)-2));

}
```



GATE 2015

What is the output of the following C code? Assume that the address of x is 2000 (in decimal) and an integer requires four bytes of memory.

```
int main () {
        unsigned int x [4] [3] =
        {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12}};
        printf ("%u, %u, %u", x + 3, *(x + 3), *(x + 2) + 3);
}
```

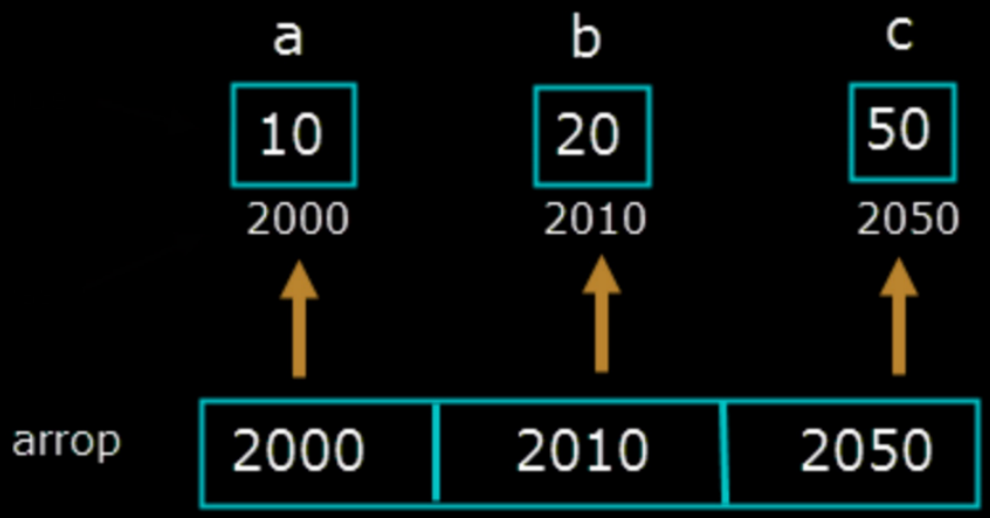
A. $2036, 2036, 2036$
B. $2012, 4, 2204$
C. $2036, 10, 10$
D. $2012, 4, 6$

GATE 2008

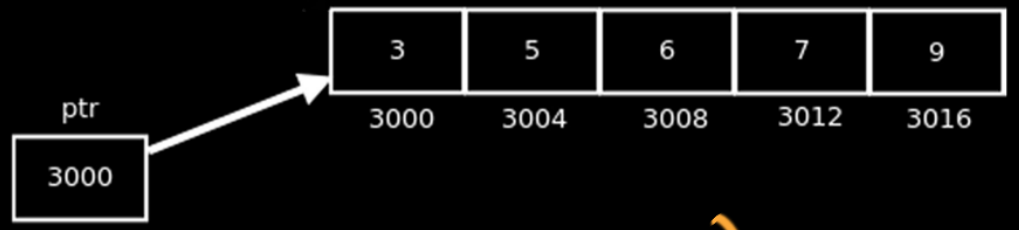
```
int main ()
{
        int i, j;
        char a [2] [3] = {{'a', 'b', 'c'}, {'d', 'e', 'f'}};
        char b [3] [2];
        char *p = *b;


        for (i = 0; i < 2; i++) {

              for (j = 0; j < 3; j++) {

                   *(p + 2*j + i) = a [i] [j];

              }
        }
}
```

Array of pointers vs pointer to array

Array of pointers vs pointer to array



int *arrop[3]



int (*ptr)[5]

GO CLASSES

Pointer to array

int (*ptr) [5] ;

```
int(*p)[4];
```

↳ p is a pointer to an integer array of size 4.

```
int *p[4];
```

↳ p is an array of 4 integer pointers

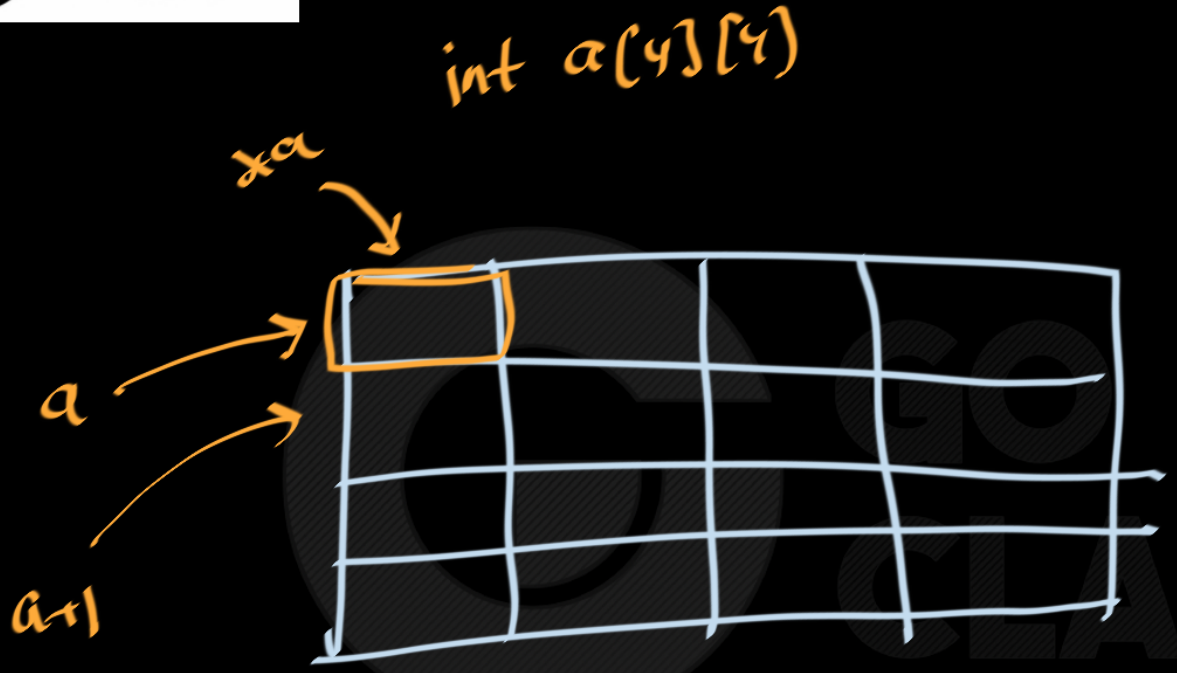
```
int(*p)[4];
```

```
int *p[4];
```

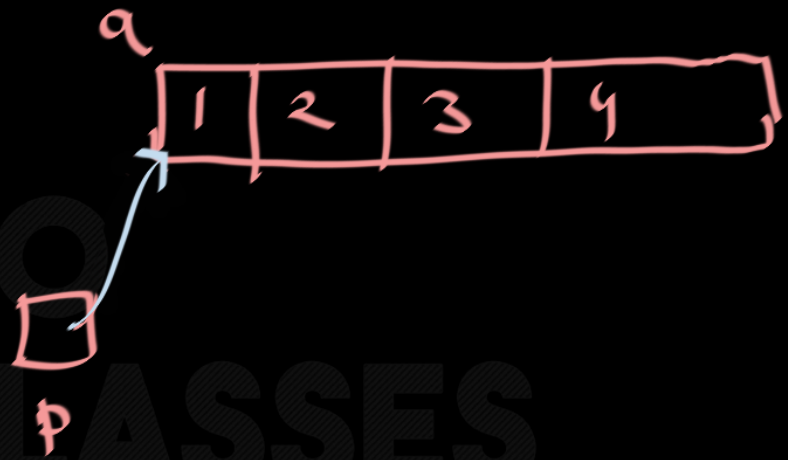
p is a pointer to an integer array of size 4.

```
int *t;
```

t is an integer pointer


int a[4][4]
xa
a
a+1

```c
#include <stdio.h>
int main()
{
    int(*p)[4];
    int a[4] = {1,2,3,4};
    p = &a;
    for (int i = 0; i < 4; ++i)
        printf("%d\n", *(*p + i));

    return 0;
}
```

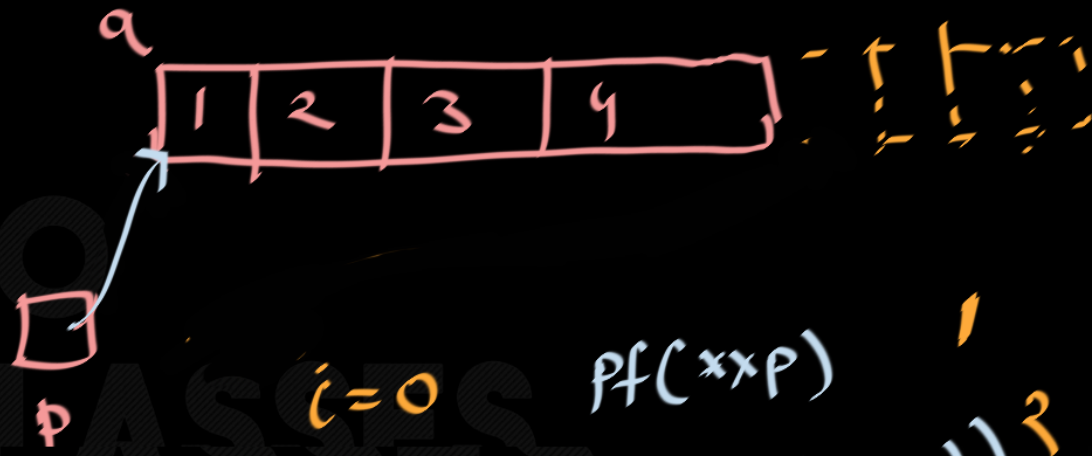


P[0][i]

an address of entire array

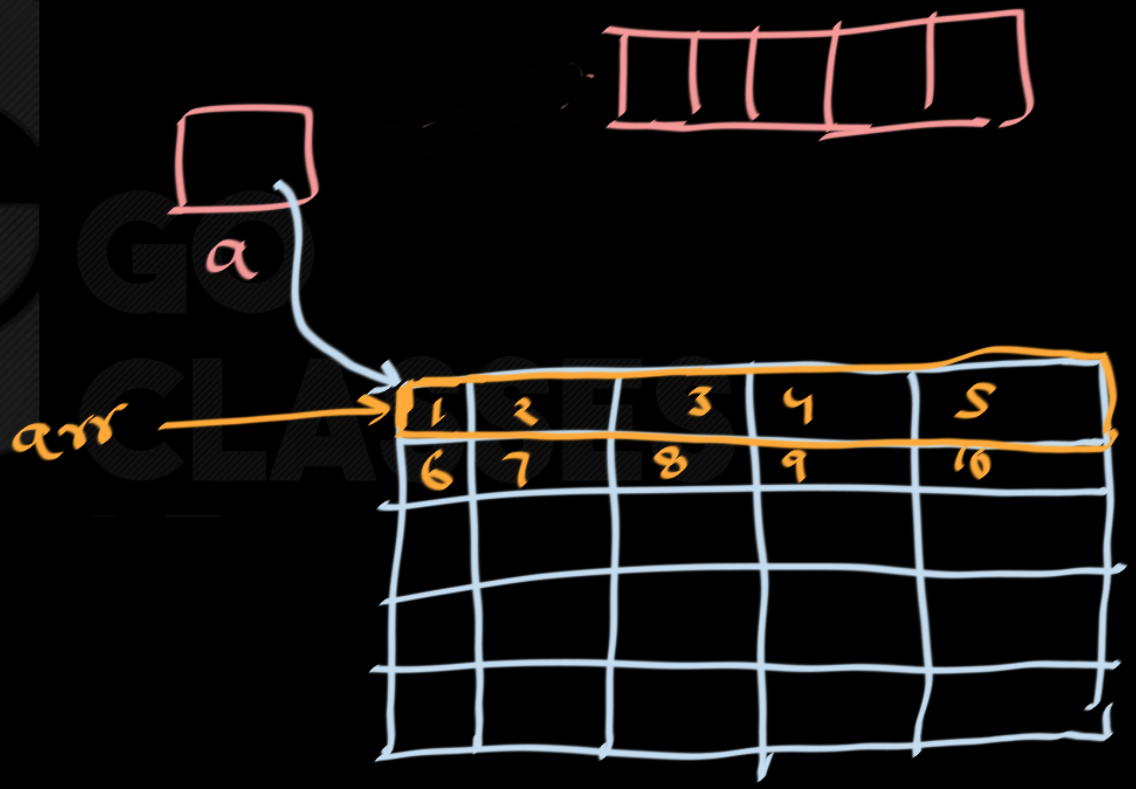
&a → Pointer to entire array

```c
#include <stdio.h>
int main()
{
    int(*p)[4];
    int a[4] = {1,2,3,4};
    p = &a;
    for (int i = 0; i < 4; ++i)
        printf("%d\n", *(*p + i));

    return 0;
}
```

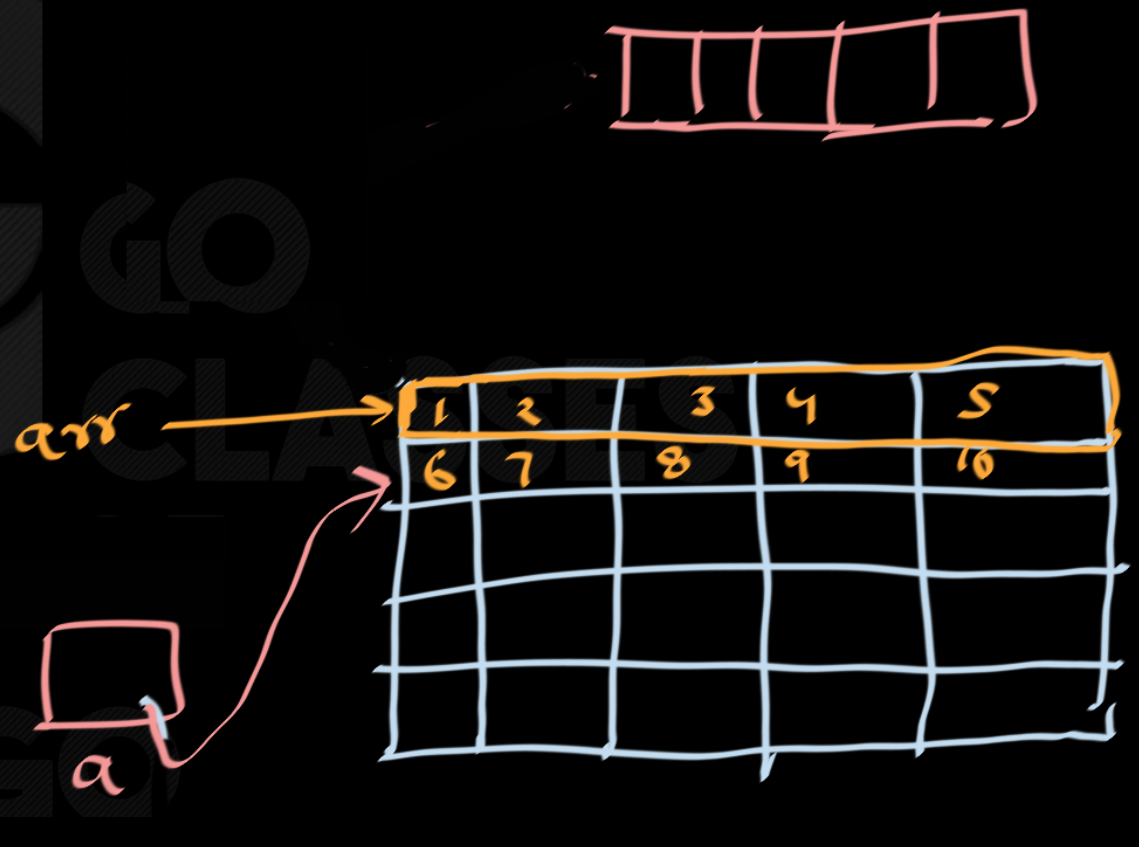


i=0 Pf(**p) 1
i=1 Pf(*(*p+1)) 2
i=2 Pf(*(*p+2)) 3
i=3 Pf(*(*p+3)) 4

```c
int main()
{
    int (*a)[5];
    int arr[5][5]={
                {1,2,3,4,5},
                {6,7,8,9,10}
                };

    a = arr;
    ++a ;

    printf("%d", **a);
}
```

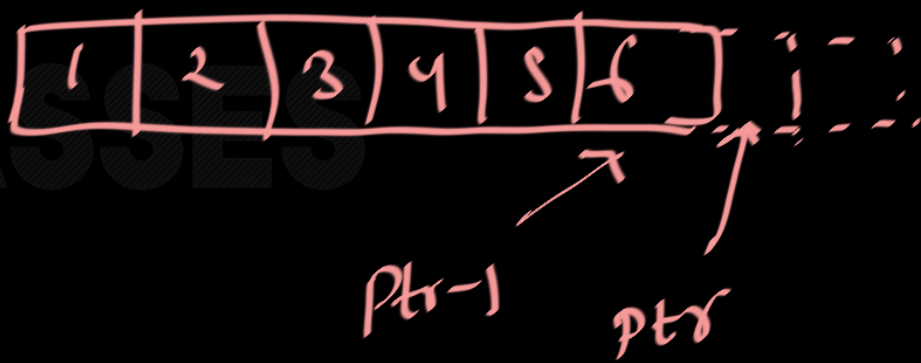


```c
int main()
{
    int (*a)[5];
    int arr[5][5]={
                {1,2,3,4,5},
                {6,7,8,9,10}
                };

    a = arr;
    ++a ;

    printf("%d", **a);
}
```



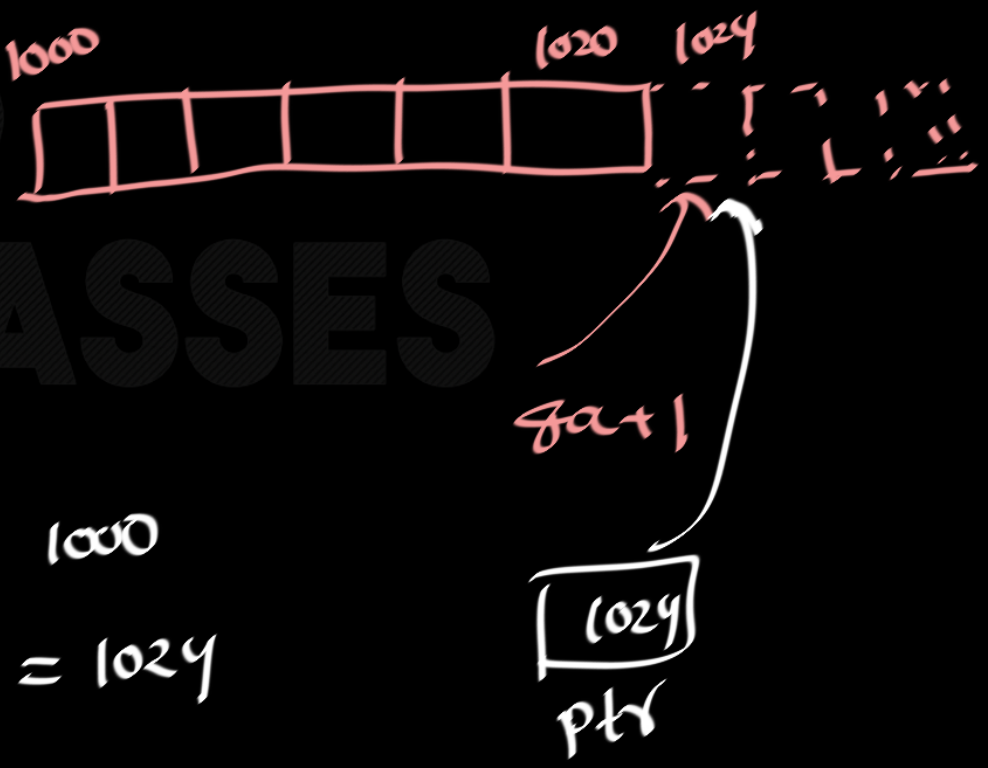
```c
#include<stdio.h>

int main()
{

  int a[] = {1, 2, 3, 4, 5, 6};

  int *ptr = (int*) (&a+1);

  printf("%d ", *(ptr-1) );

  return 0;
}
```

→ 6



```c
#include<stdio.h>

int main()
{

  int a[] = {1, 2, 3, 4, 5, 6};

  int *ptr = &a+1;


  printf("%d ", *(ptr-1) );

  return 0;
}
```

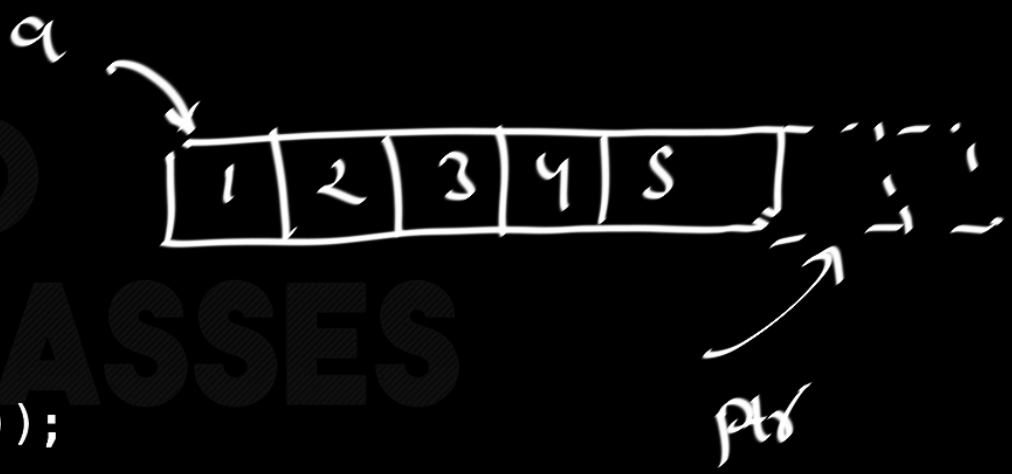
→ 6



$\&a \equiv 1000$

$\&a+1 = 1024$

```c
#include <stdio.h>

int main() {
    int a[] = {1, 2, 3, 4, 5};
    int *ptr = (int*)(&a + 1);

    printf("%d %d", *(a + 1), *(ptr - 1));

    return 0;
}
```

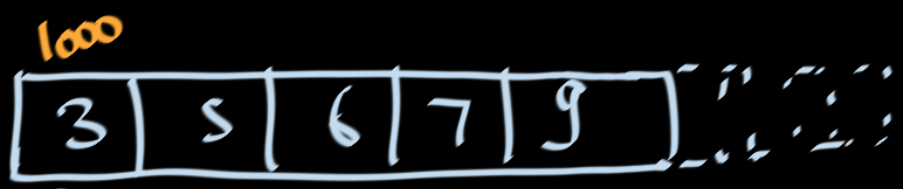


2 5

8 Byte address

```c
#include <stdio.h>
int main() {
    int arr[] = {3, 5, 6, 7, 9};
    int *p = arr;
    int (*ptr)[5] = &arr;

    printf("p = %d\n", *p);
    printf("ptr[0][3] = %d\n", ptr[0][3]);

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

    printf("sizeof(ptr) = %d\n", sizeof(ptr));
    printf("sizeof(*ptr) = %d\n", sizeof(*ptr));

    return 0;
}
```

arr



p

ptr

3

7

8

4

8

5x4 = 20

optional

p ≡ 1000 ptr ≡ 1004

ptr ≡ 1000 ptr+1 ≡ 1020

arr ≡ 1000 arr+1 ≡ 1004

GO CLASSES

`int *t` — t : is a pointer to int

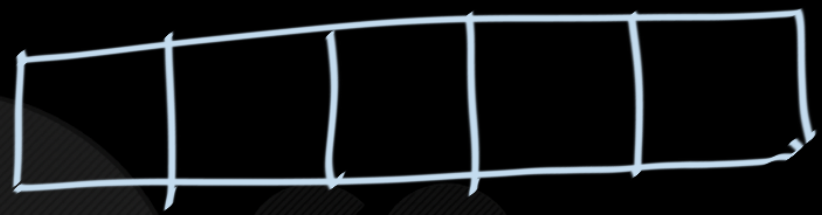
*t : is a integer

`int (*p)[4]` — p: is a pointo to int array of size 4

*p : is an int array of size 4

GO CLASSES

int (*ptr) [5];



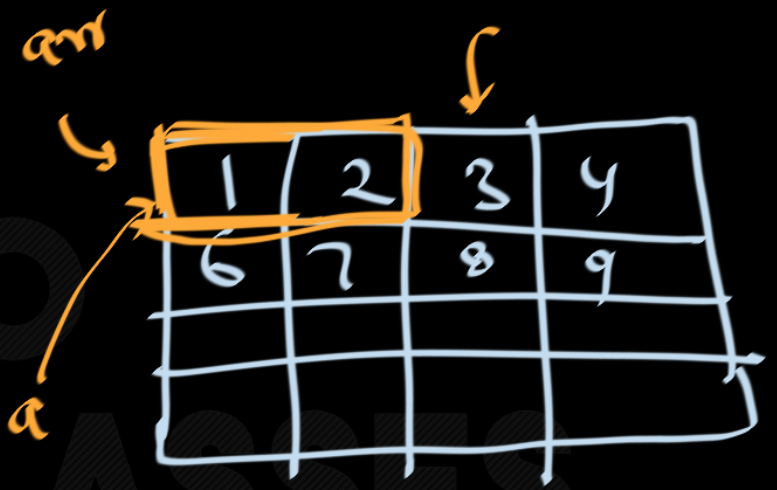
ptr

*ptr = x

*ptr ≡ one D array

```c
#include<stdio.h>

int main()
{
    int (*a)[2];
    int arr[4][4]={
                1,2,3,4,
                6,7,8,9
                };

    a = arr;
    ++a ;

    printf("%d", **a);
}
```

↳3



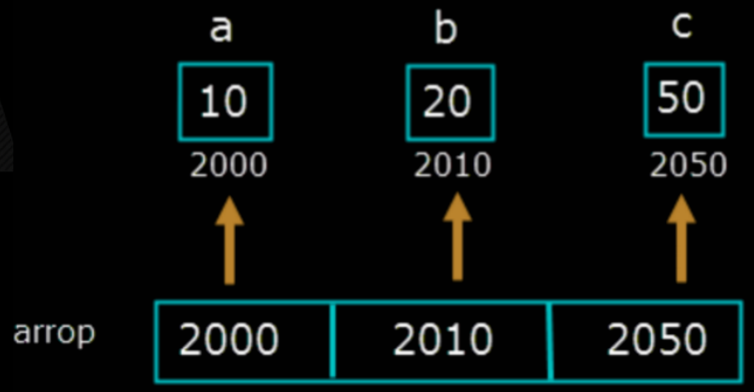
GO CLASSES

Array of Pointers

```
int *arrop[3];
```

or

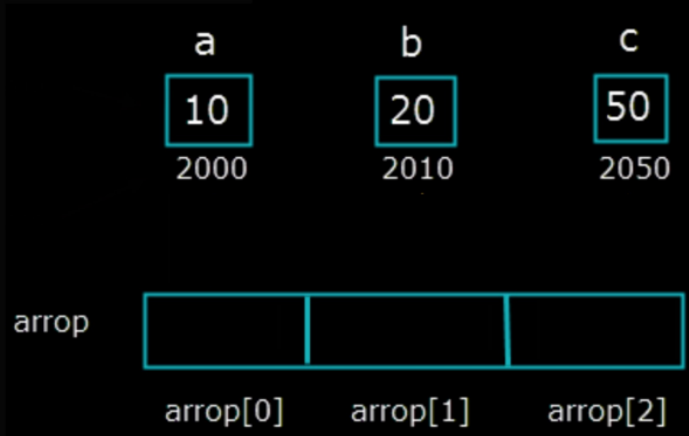
```
int *(arrop[3]);
```



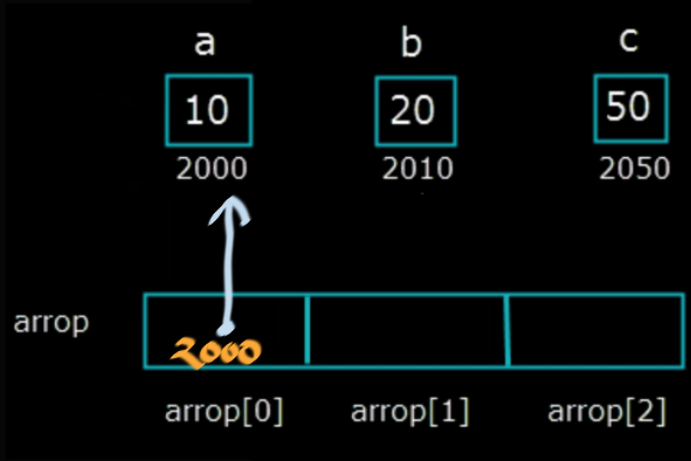
Array of Pointers

```
int *array[10]; // array of 10 int pointers
```

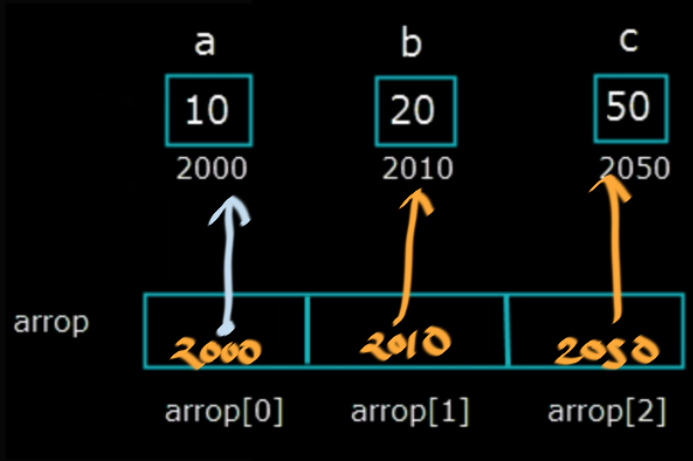
```c
int main()
{
    int *arrop[3];
    int a = 10, b = 20, c = 50, i;

    arrop[0] = &a;
    arrop[1] = &b;
    arrop[2] = &c;

    for(i = 0; i < 3; i++)
    {
        printf("Address = %d\t Value = %d\n", arrop[i], *arrop[i]);
    }

    return 0;
}
```



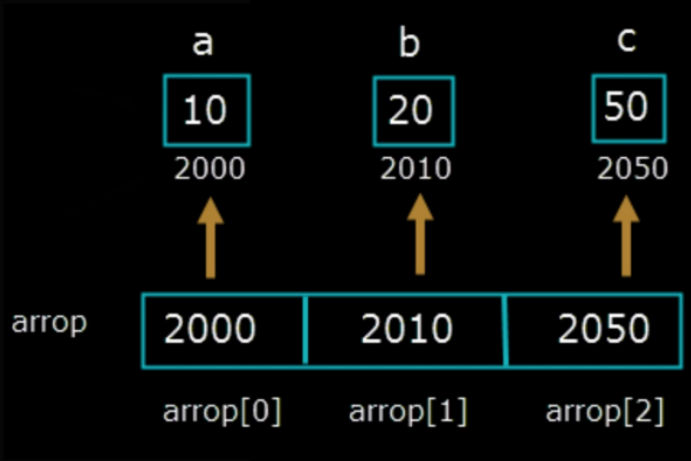
```c
int main()
{
    int *arrop[3];
    int a = 10, b = 20, c = 50, i;

    arrop[0] = &a;
    arrop[1] = &b;
    arrop[2] = &c;

    for(i = 0; i < 3; i++)
    {
        printf("Address = %d\t Value = %d\n", arrop[i], *arrop[i]);
    }

    return 0;
}
```



```c
int main()
{
    int *arrop[3];
    int a = 10, b = 20, c = 50, i;

    arrop[0] = &a;
    arrop[1] = &b;
    arrop[2] = &c;

    for(i = 0; i < 3; i++)
    {
        printf("Address = %d\t Value = %d\n", arrop[i], *arrop[i]);
    }

    return 0;
}
```



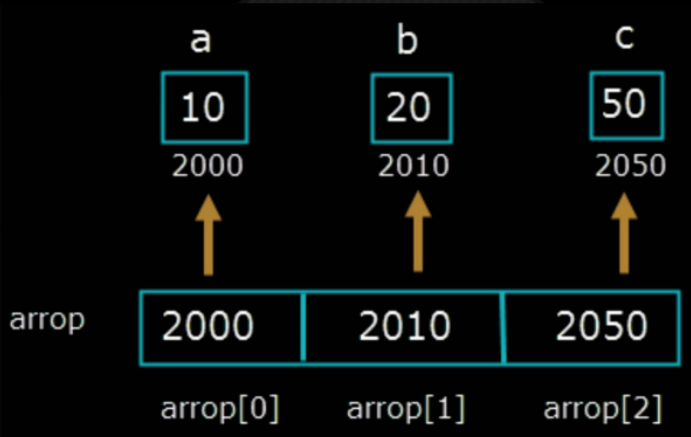
```c
int main()
{
    int *arrop[3];
    int a = 10, b = 20, c = 50, i;

    arrop[0] = &a;
    arrop[1] = &b;
    arrop[2] = &c;

    for(i = 0; i < 3; i++)
    {
        printf("Address = %p\t Value = %d\n", arrop[i], *arrop[i]);
    }

    return 0;
}
```



2000 10 *arrop[0]
2010 20 *2000
2050 50 10

GO CLASSES

int * arrop[3]



→ b ✓ ≡ 20

*(arrop[1]) ≡ 20

*arrop[1] ≡ 20 ← Preferred

*(arrop + 1) ≡ 2010

✓ arrop[1][0] ≡ 20

** (arrop + 1) ≡ 20

GO CLASSES

char a, b, c;

a = 'p'

b = 'z'

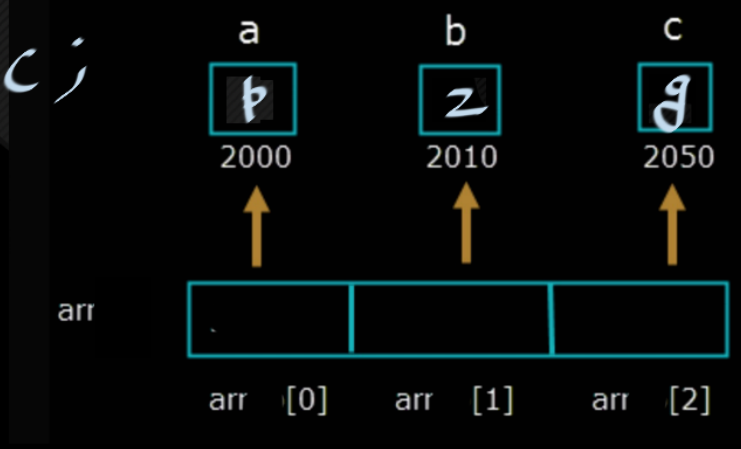
c = 'g'

char *arr[3];

arr[0] = &a;

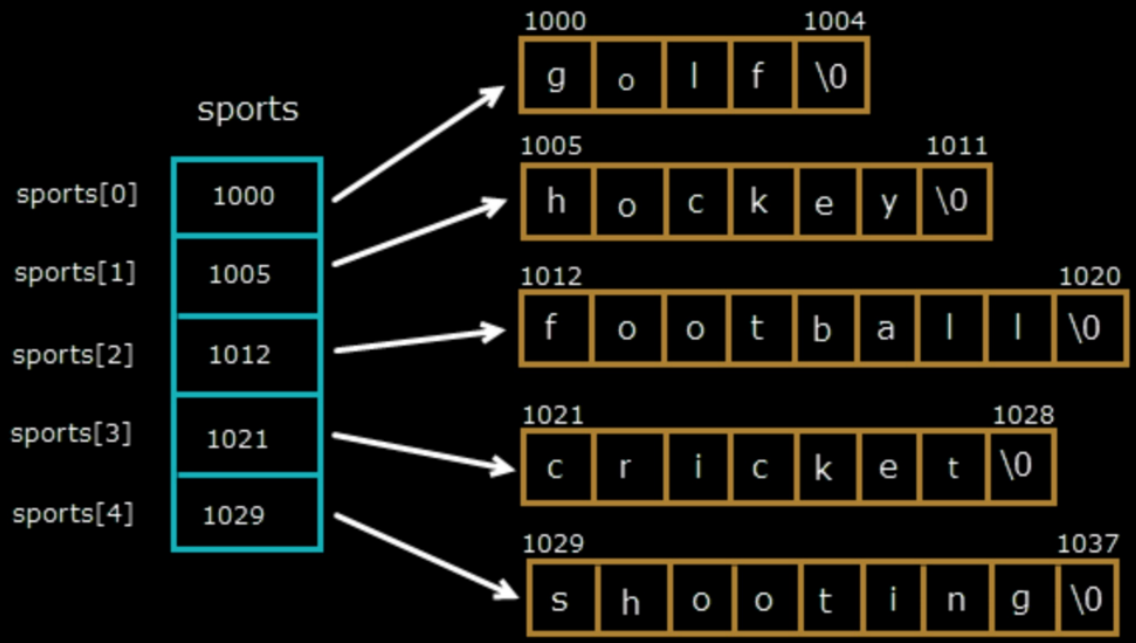
arr[1] = &b;

arr[2] = &c;



arr is an array of char pointers

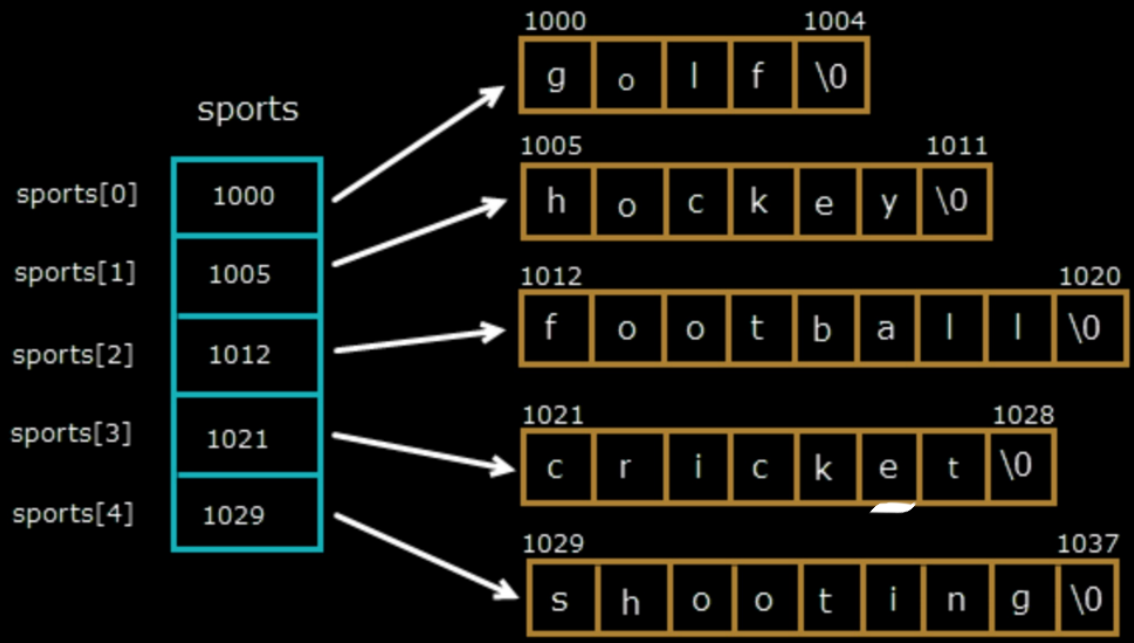
```
char *sports[5] = {
                    "golf",
                    "hockey",
                    "football",
                    "cricket",
                    "shooting"
                  };
```




```
sports[1][0]  = ?h
```

```
char *sports[5] = {
                    "golf",
                    "hockey",
                    "football",
                    "cricket",
                    "shooting"
                  };
```

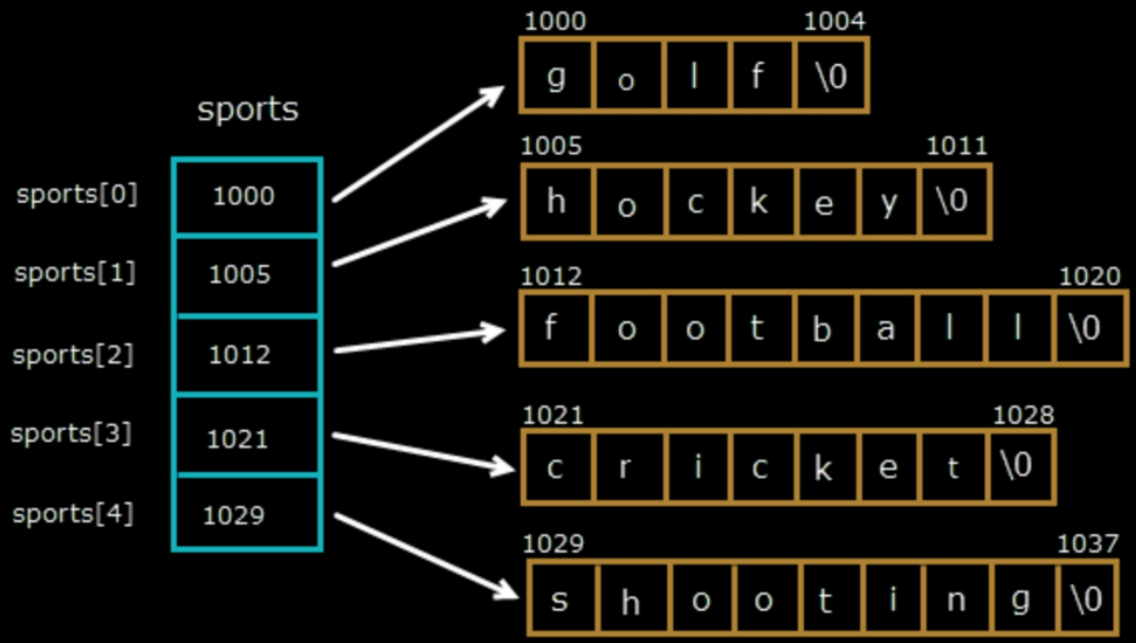
sports[1][0] = h



sports[2][3] = t

sports[3] + 5 = ? 1026

```
char *sports[5] = {
                    "golf",
                    "hockey",
                    "football",
                    "cricket",
                    "shooting"
                  };
```

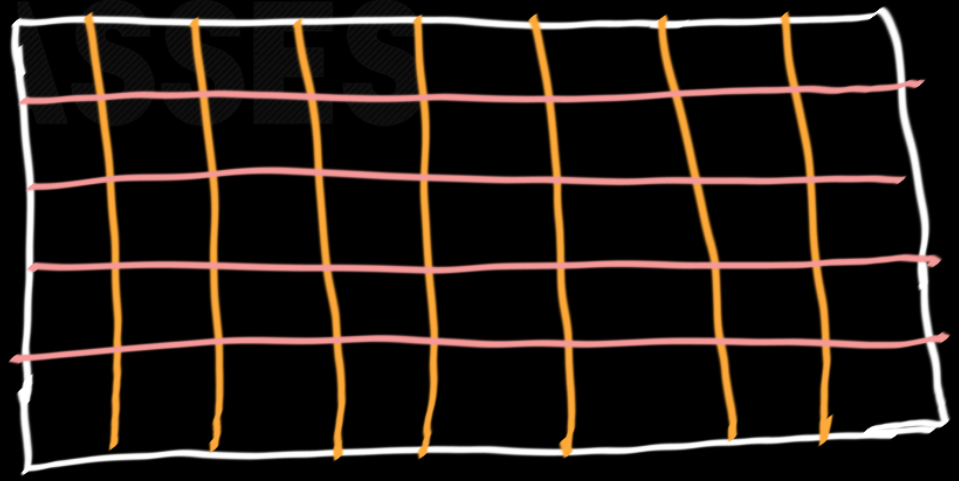



```
sports[1][0]  = 't';
```

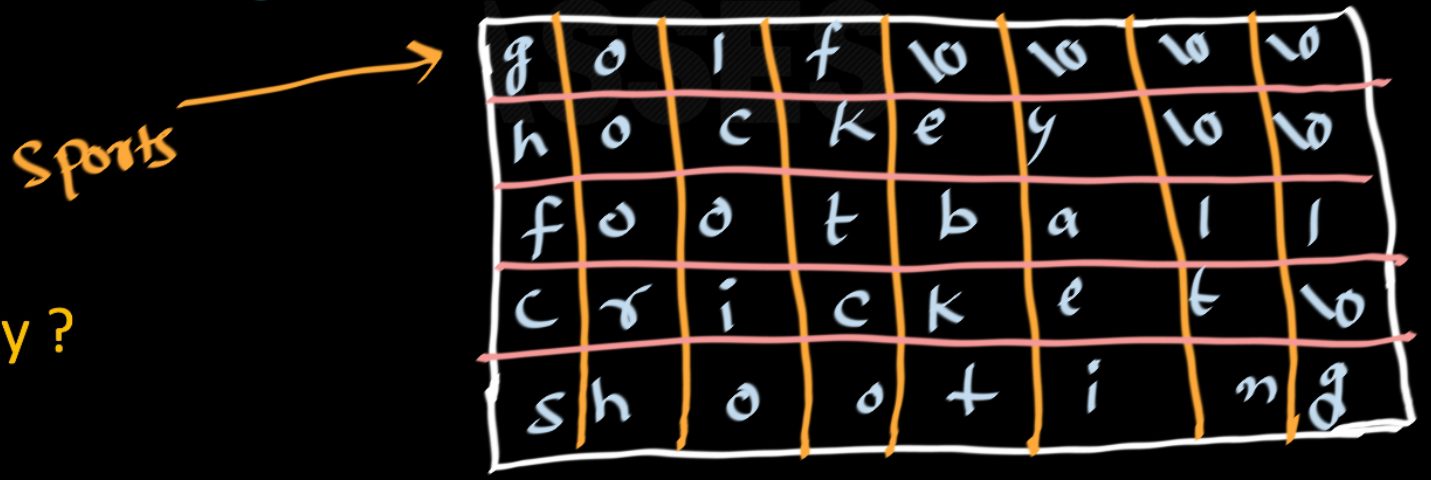
will this work ? NO bcoz of string literals

```
char sports[5][8] = {
                        "golf",
                        "hockey",
                        "football",
                        "cricket",
                        "shooting"
                    };
```

How will it look pictorially ?



```
char sports[5][8] = {
                        "golf",
                        "hockey",
                        "football",
                        "cricket",
                        "shooting"
                    };
```



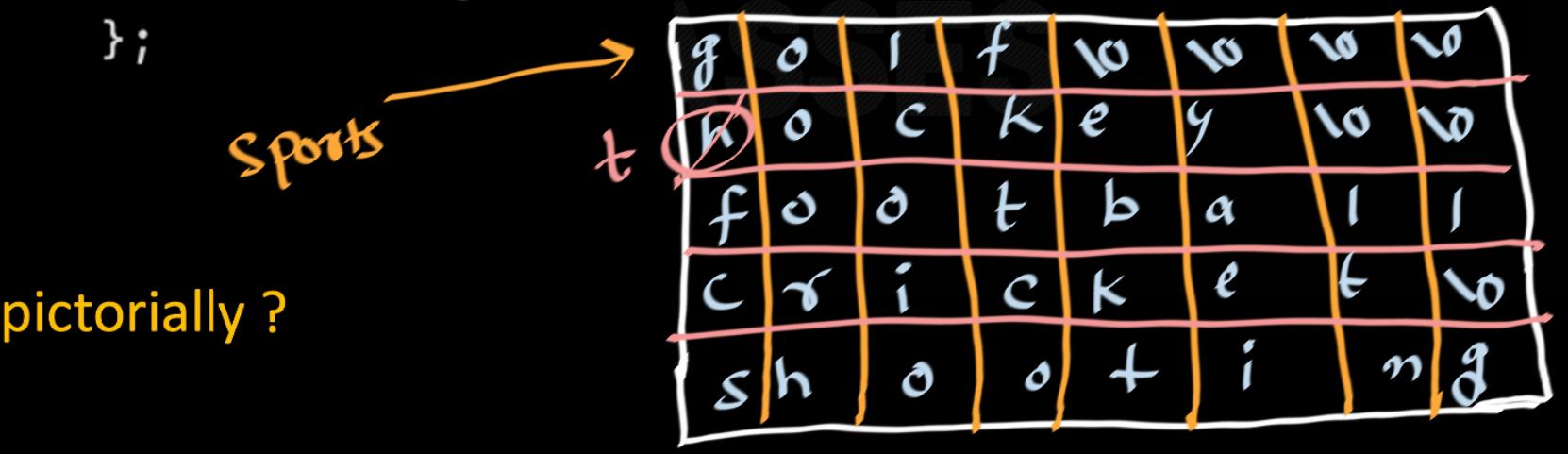
How will it look pictorially ?

```
char sports[5][8] = {
                        "golf",
                        "hockey",
                        "football",
                        "cricket",
                        "shooting"
                    };
```

```
sports[1][0]  = 't';
```

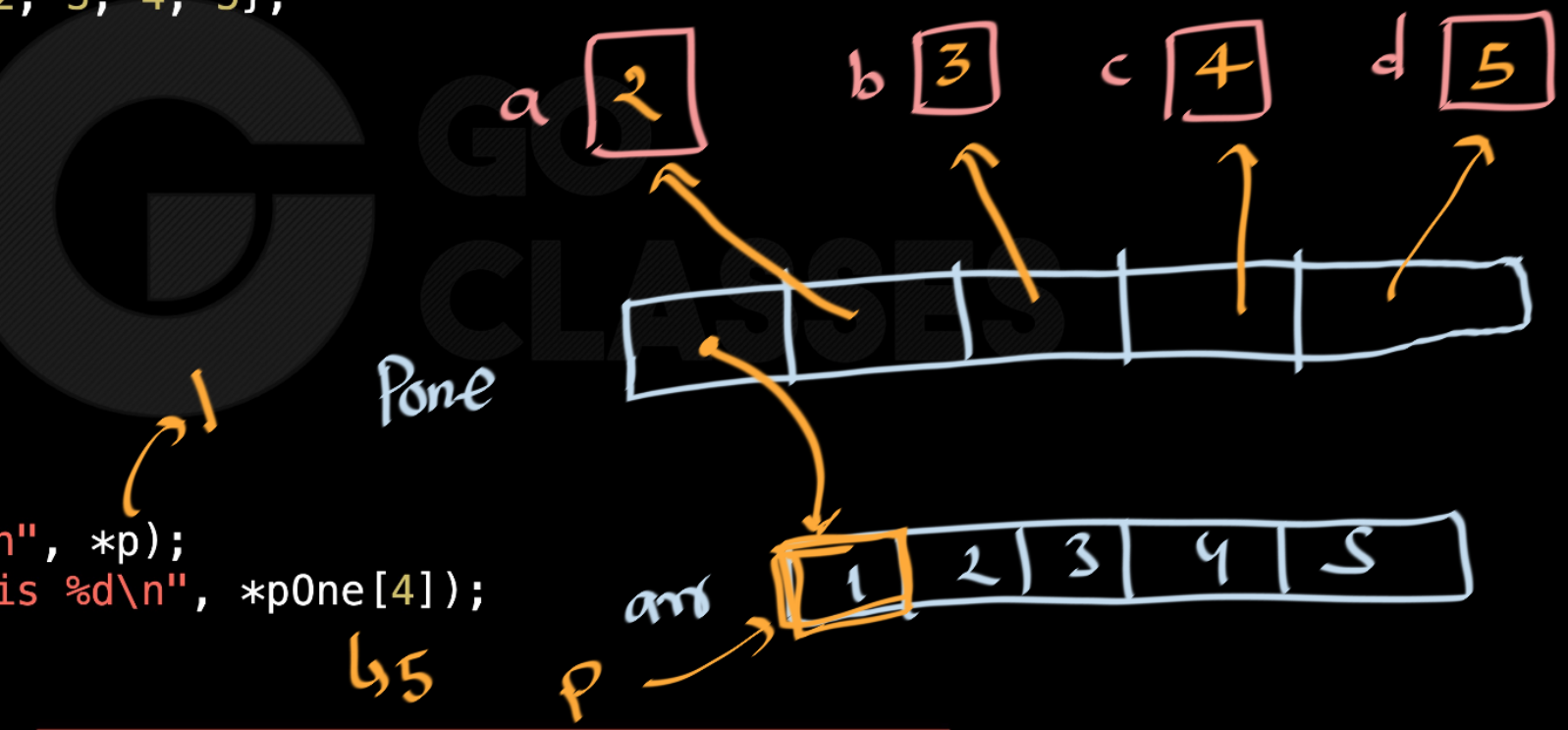
will this work ?

Yes

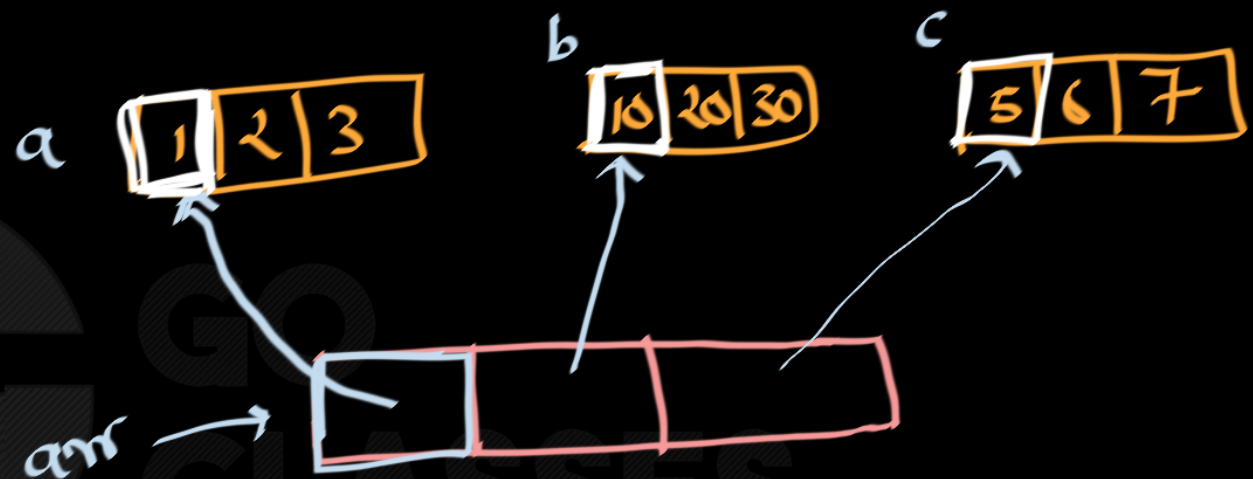


How will it look pictorially ?

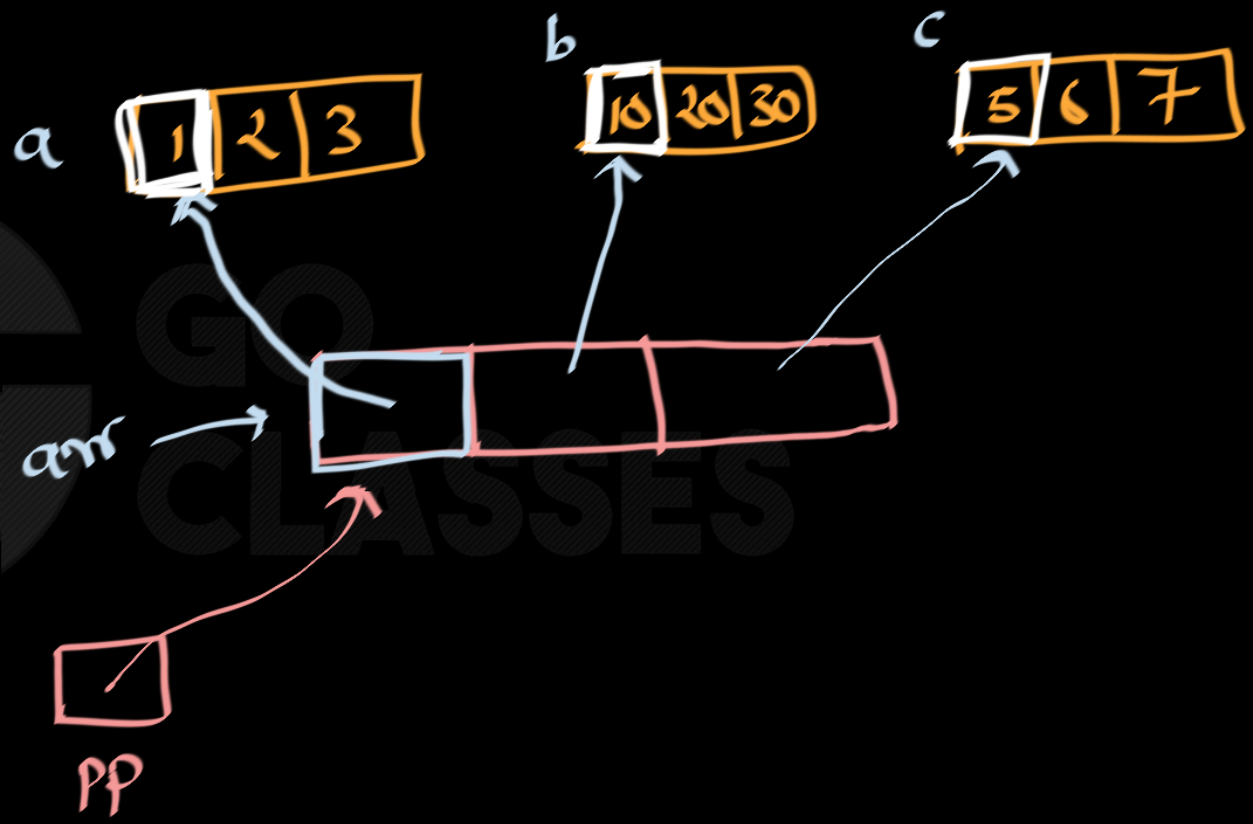
```c
#include <stdio.h>

int main() {
    int a = 2, b = 3, c = 4, d = 5;
    int arr[5] = {1, 2, 3, 4, 5};

    int *pOne[5];
    int *p = arr;

    pOne[0] = arr;
    pOne[1] = &a;
    pOne[2] = &b;
    pOne[3] = &c;
    pOne[4] = &d;

    printf("*p is %d\n", *p);
    printf("*pOne[4] is %d\n", *pOne[4]);

    return 0;
}
```



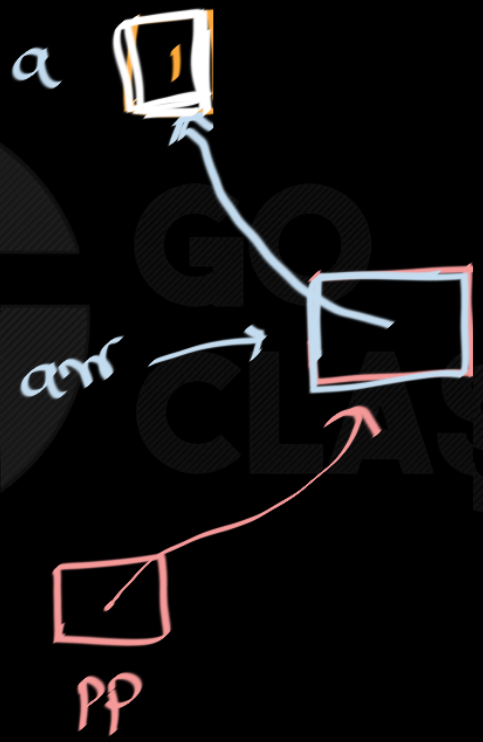
```c
#include<stdio.h>

int main()
{
    int a[] = {1, 2, 3};
    int b[] = {10, 20, 30};
    int c[] = {5, 6, 7};

    int *arr[] = {a, b, c};









    return 0;
}
```



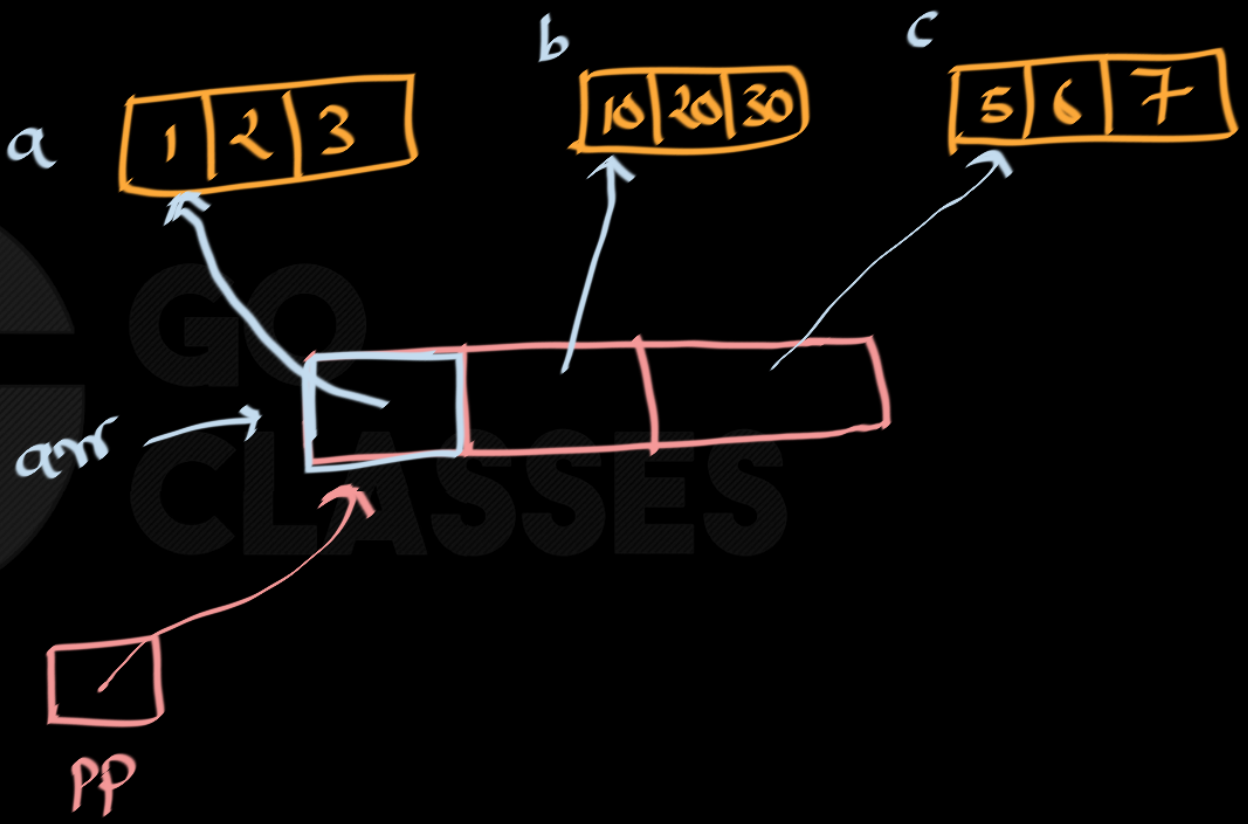
```c
#include<stdio.h>

int main()
{
    int a[] = {1, 2, 3};
    int b[] = {10, 20, 30};
    int c[] = {5, 6, 7};

    int *arr[] = {a, b, c};



    int **pp = arr;



    return 0;
}
```



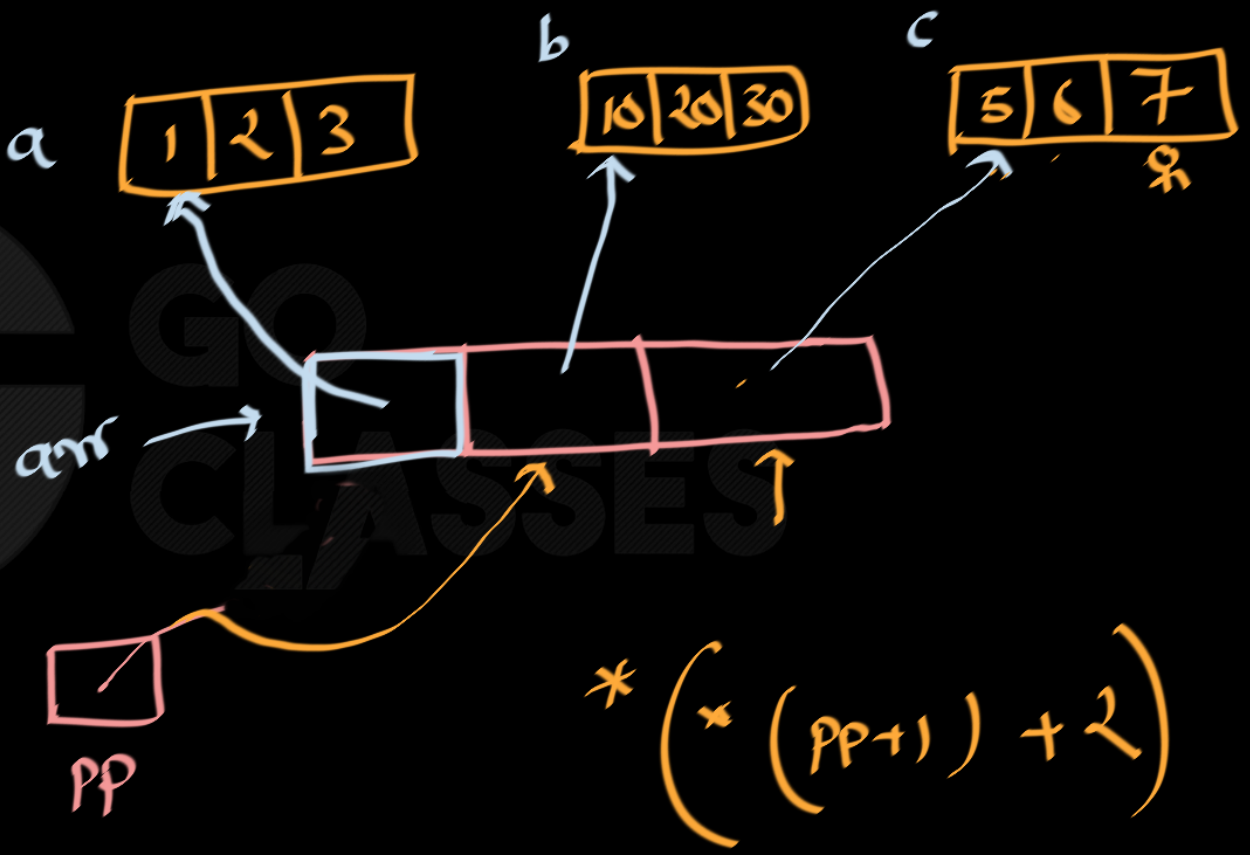
```c
#include<stdio.h>

int main()
{
    int a[] = {1, 2, 3};
    int b[] = {10, 20, 30};
    int c[] = {5, 6, 7};

    int *arr[] = {a, b, c};



    int **pp = arr;



    return 0;
}
```



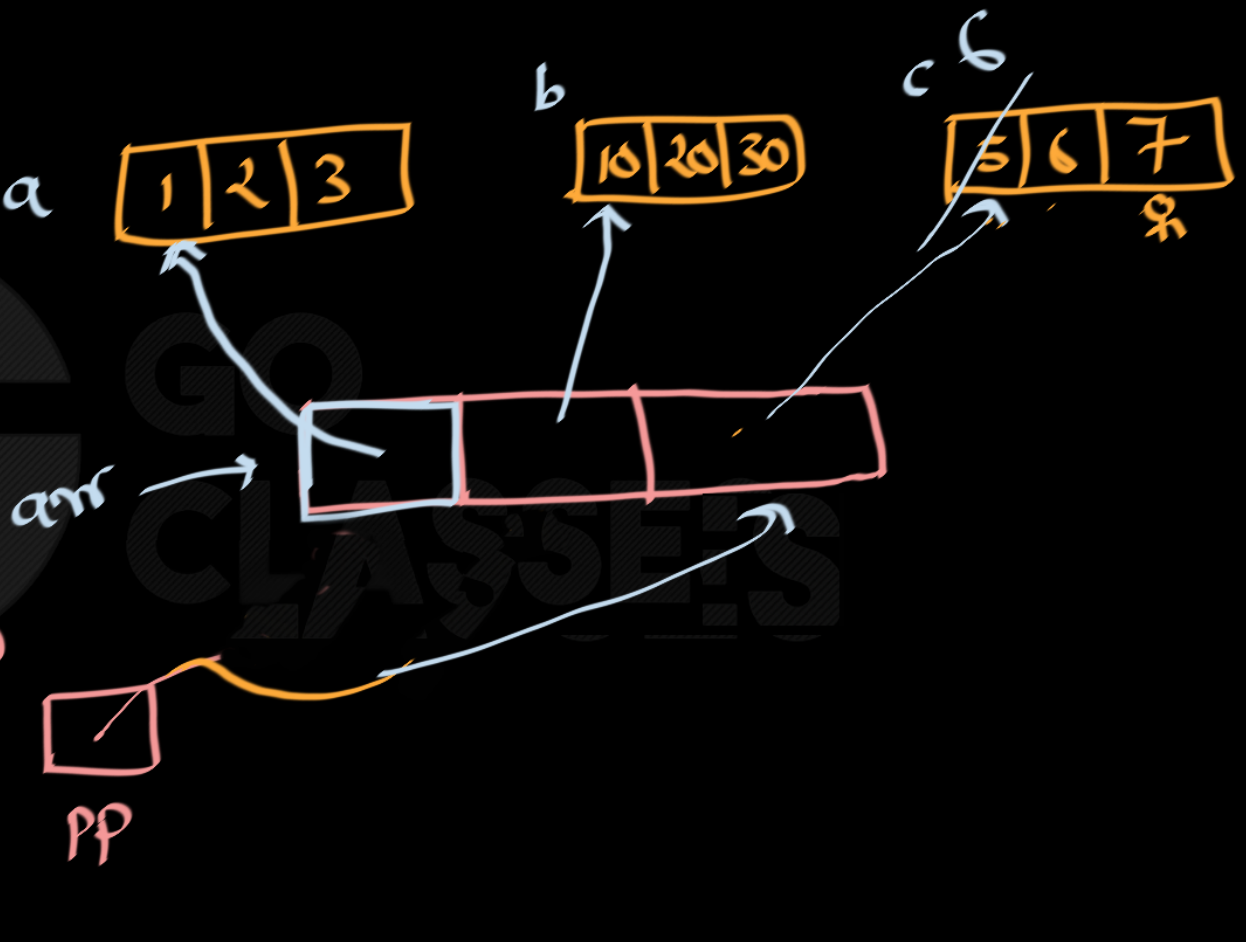
```c
#include<stdio.h>

int main()
{
    int a[] = {1, 2, 3};
    int b[] = {10, 20, 30};
    int c[] = {5, 6, 7};

    int *arr[] = {a, b, c};

    int **pp = arr;

    pp++;

    printf("%d\n", pp[1][2]);

    printf("%d\n", **(pp++));

    printf("%d\n", (**pp)++);

    printf("%d\n", **pp);

    return 0;
}
```



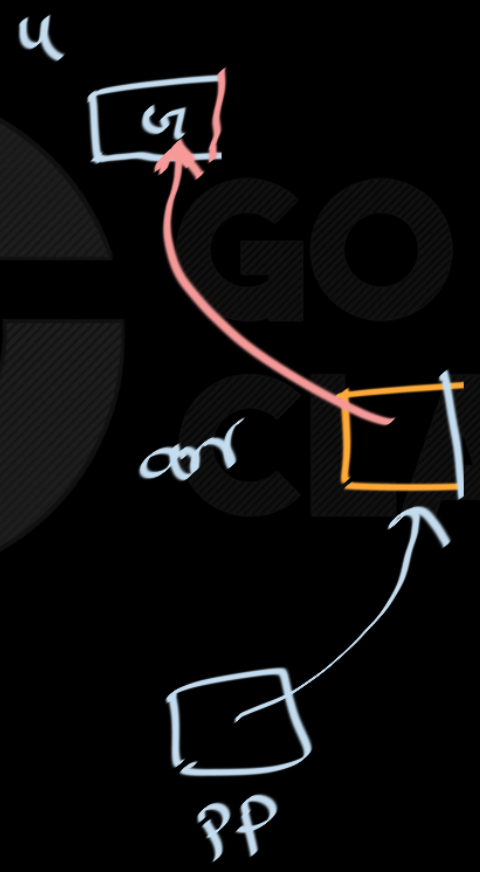
```c
#include<stdio.h>

int main()
{
    int a[] = {1, 2, 3};
    int b[] = {10, 20, 30};
    int c[] = {5, 6, 7};

    int *arr[] = {a, b, c};

    int **pp = arr;

    pp++;

    printf("%d\n", pp[1][2]);

    printf("%d\n", **(pp++));

    printf("%d\n", (**pp)++);

    printf("%d\n", **pp);

    return 0;
}
```



```c
#include<stdio.h>

int main()
{
    int a[] = {1, 2, 3};
    int b[] = {10, 20, 30};
    int c[] = {5, 6, 7};

    int *arr[] = {a, b, c};

    int **pp = arr;

    pp++;

    printf("%d\n", pp[1][2]);

    printf("%d\n", **(pp++));

    printf("%d\n", (**pp)++);

    printf("%d\n", **pp);

    return 0;
}
```

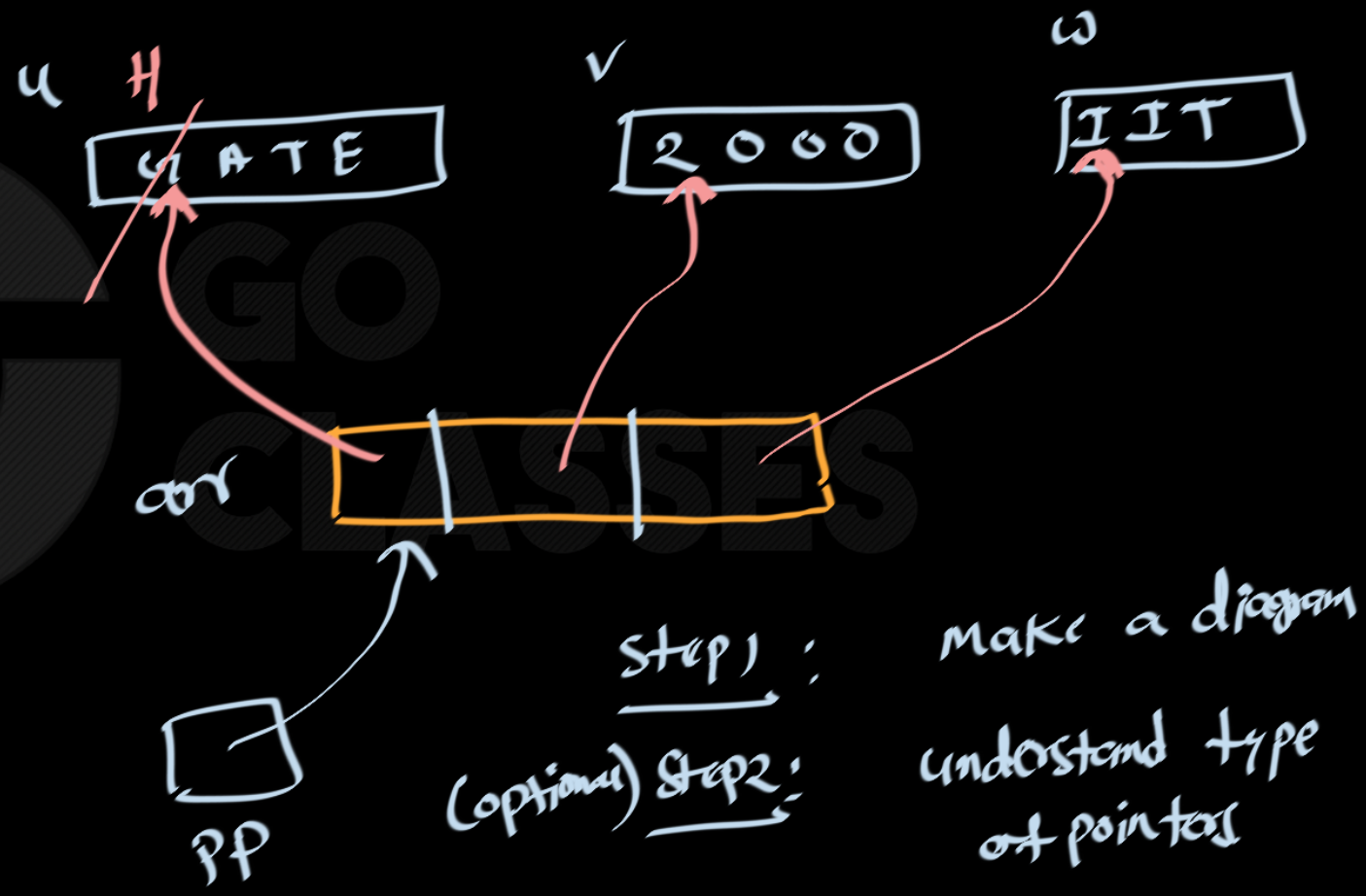


```c
#include<stdio.h>

int main()
{
    char u[] = "GATE";
    char v[] = "2000";
    char w[] = "IIT";

    char *arr[] = {u, v, w};

    char **pp = arr;

    printf("%c\n", (**pp)++);

    printf("%c\n", **(pp++));

    printf("%s\n", *pp);

    return 0;
}
```



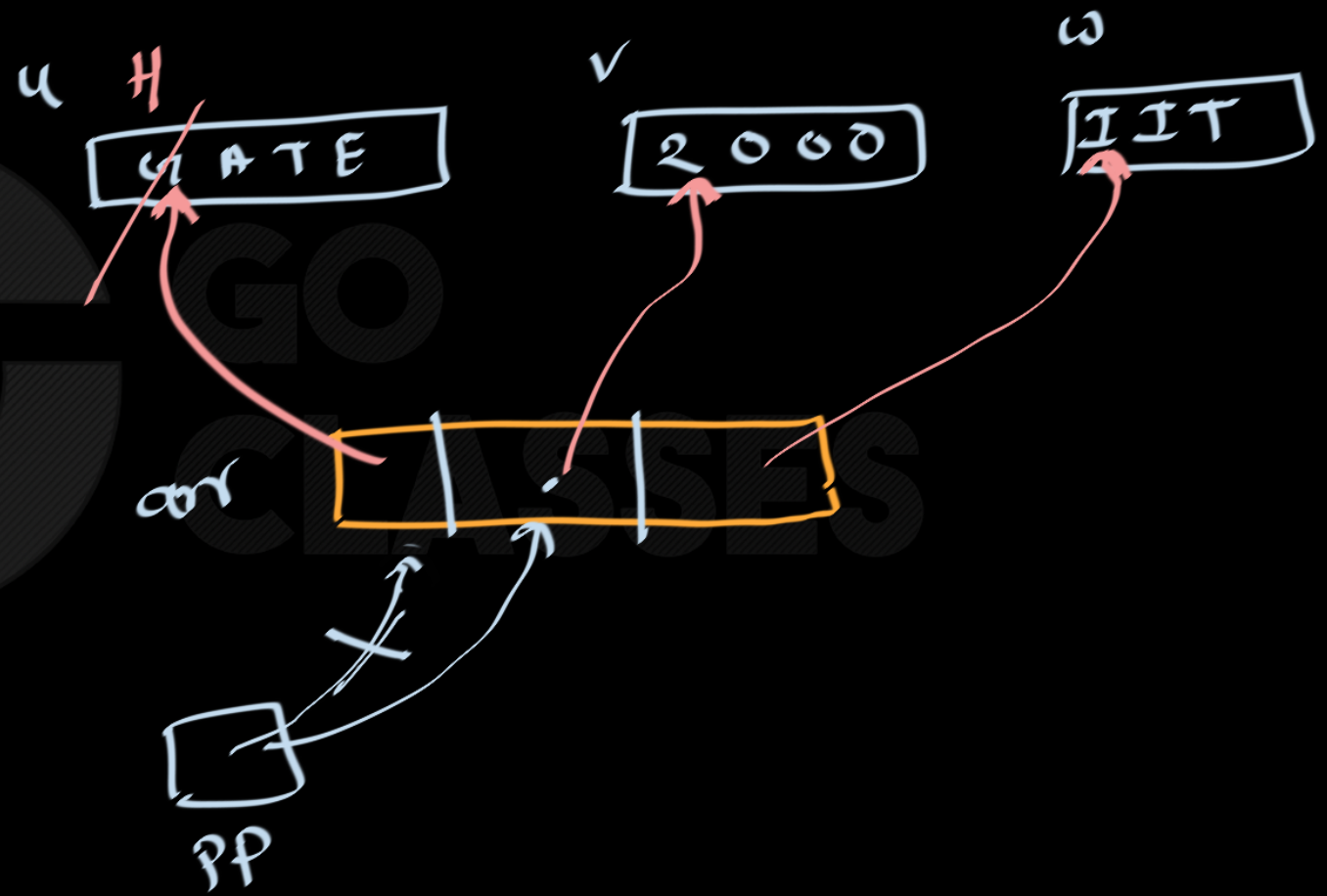
GO CLASSES

```c
#include<stdio.h>

int main()
{
    char u[] = "GATE";
    char v[] = "2000";
    char w[] = "IIT";

    char *arr[] = {u, v, w};

    char **pp = arr;
    printf("%c\n", (**pp)++);

    printf("%c\n", **(pp++));

    printf("%s\n", *pp);

    return 0;
}
```



GO CLASSES

```c
#include<stdio.h>

int main()
{
    char u[] = "GATE";
    char v[] = "2000";
    char w[] = "IIT";

    char *arr[] = {u, v, w};

    char **pp = arr;

    printf("%c\n", (**pp)++);

    printf("%c\n", **(pp++));

    printf("%s\n", *pp);

    return 0;
}
```



GO CLASSES

```c
#include<stdio.h>

int main()
{
    char u[] = "GATE";
    char v[] = "2000";
    char w[] = "IIT";

    char *arr[] = {u, v, w};

    char **pp = arr;

    printf("%c\n", (**pp)++);

    printf("%c\n", **(pp++));

    printf("%s\n", *pp);

    return 0;
}
```

```
Desktop — -zsh — 80×24
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
G
H
2000
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

Passing 2D array to function

- Copies address of first element of 2D array, In case of 2D arrays, first element is first row

int a[x][y];

```
func (int (*a)[y]){
}
```

```
func(int a[][y]){
}
```

```
func(int a[x][y]){
}
```

```
func(int **a)  { /* WRONG */
}
```

```
func(int a[x][]){    /* WRONG */
}
```

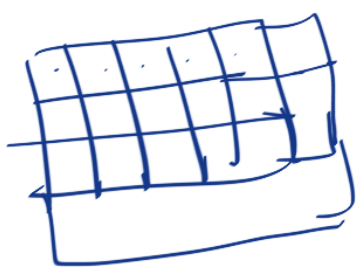
```
main ( )
{
    int a[5]
    fun ( a )
}
```

add of one integer

```
fun ( int *p )
{
}
```

```
main ( )
{
    int a[5][6]
    fun ( a )
}
```

```
fun ( int (*p)[6] )
{
}
```



add of one row of size 6

```
main ( )
{
    int a [5] [6]
    fun ( a )
}
```

```
fun ( int a[5][6] )
{
    int (*a) [6]
}
```

Convinience

internally

```
main ( )
{
    int a [5] [6]
    fun ( a )
}
```

```
fun ( int a[ ][6] )
{
}
```

internally

```
int (*a)[6]
```

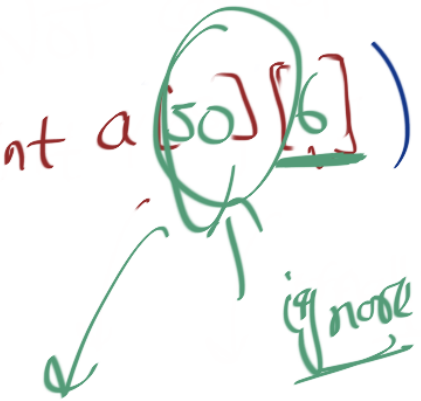
```
main ( )
{
    int a [5] [6]
    fun ( a )
}
```

```
fun ( int a[ ][ ] )
{
    int (*a) [6]
}
```

Not correct

```
main ( )
{
    int a [5] [6]
    fun ( a )
}
```

```
fun ( int a [50] [6] )
{
}
```



main ( )

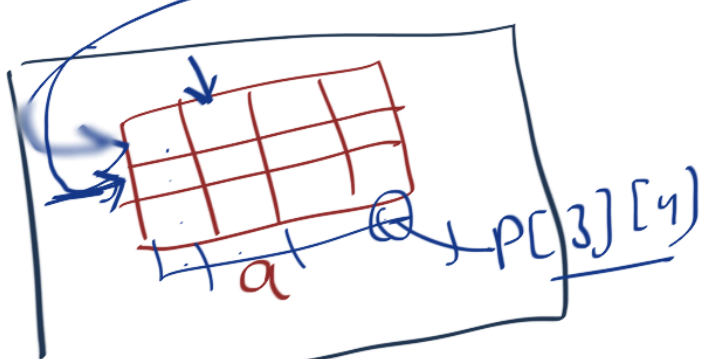
{

int a[5][6] = { 0 };

a = a+1 ✗

fun (a)

}



main

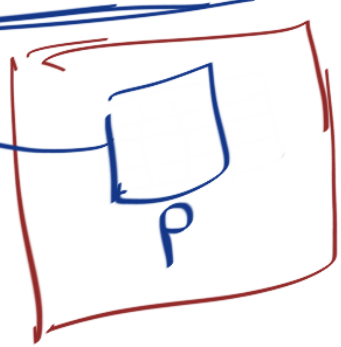
fun (int p[5][6])

{

p[3][4] = 5

p = p+1

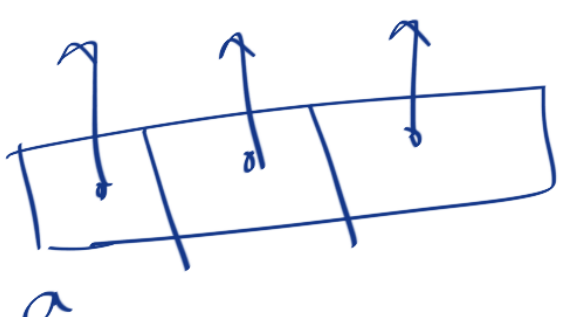
}



fun

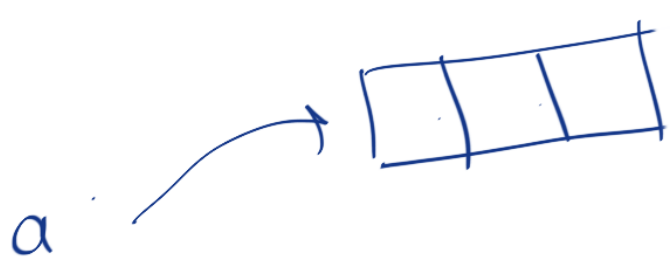
Array of pointers

`int *a[3]`



Pointer to an array

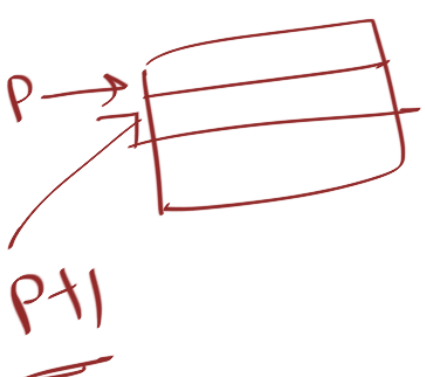
`int (*a)[3]`



2 D array

```
main ( )
{
    int a[3][4]
    fun(a)
}
```

```
fun ( int p[3][4])
{
    p ~ p is pointer
        here
}
```



2D array

```
main ( )
{
    int a[3][4]
    fun(a)
}
```

```
fun (          )
{
}
```