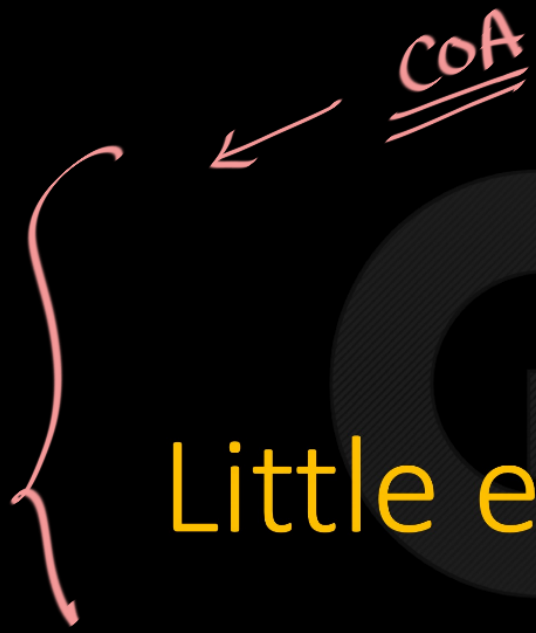


Optional

Little endian vs Big endian

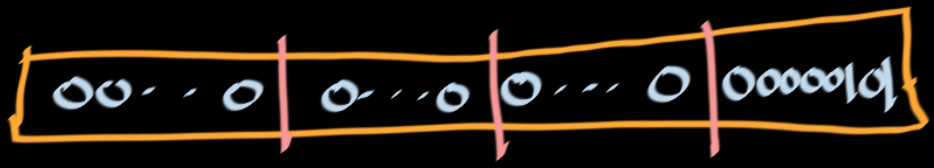
GO CLASSES

```
int a = 5;
```

5 in Binary 00.....000 101

5 in Binary 

GO CLASSES

5 in Binary 

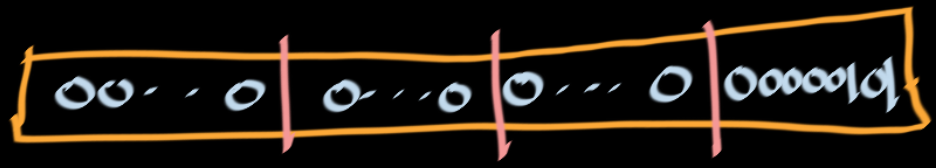
4 bytes

4! combinations X

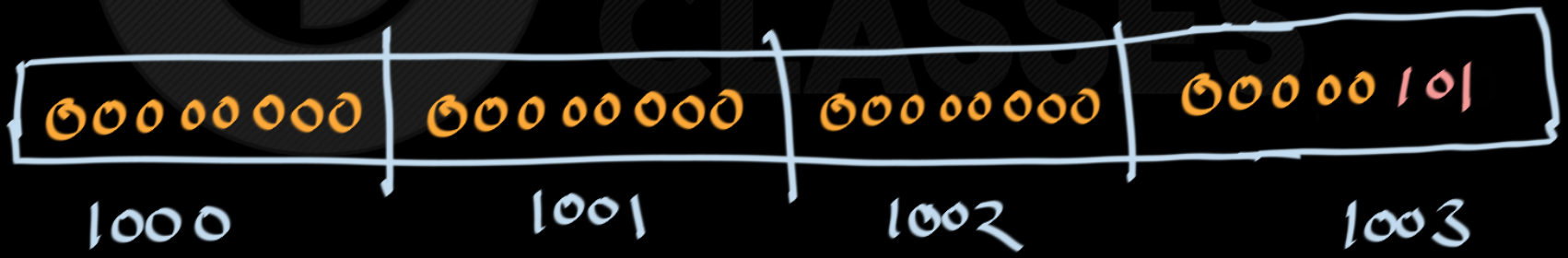
Only 2 Combination

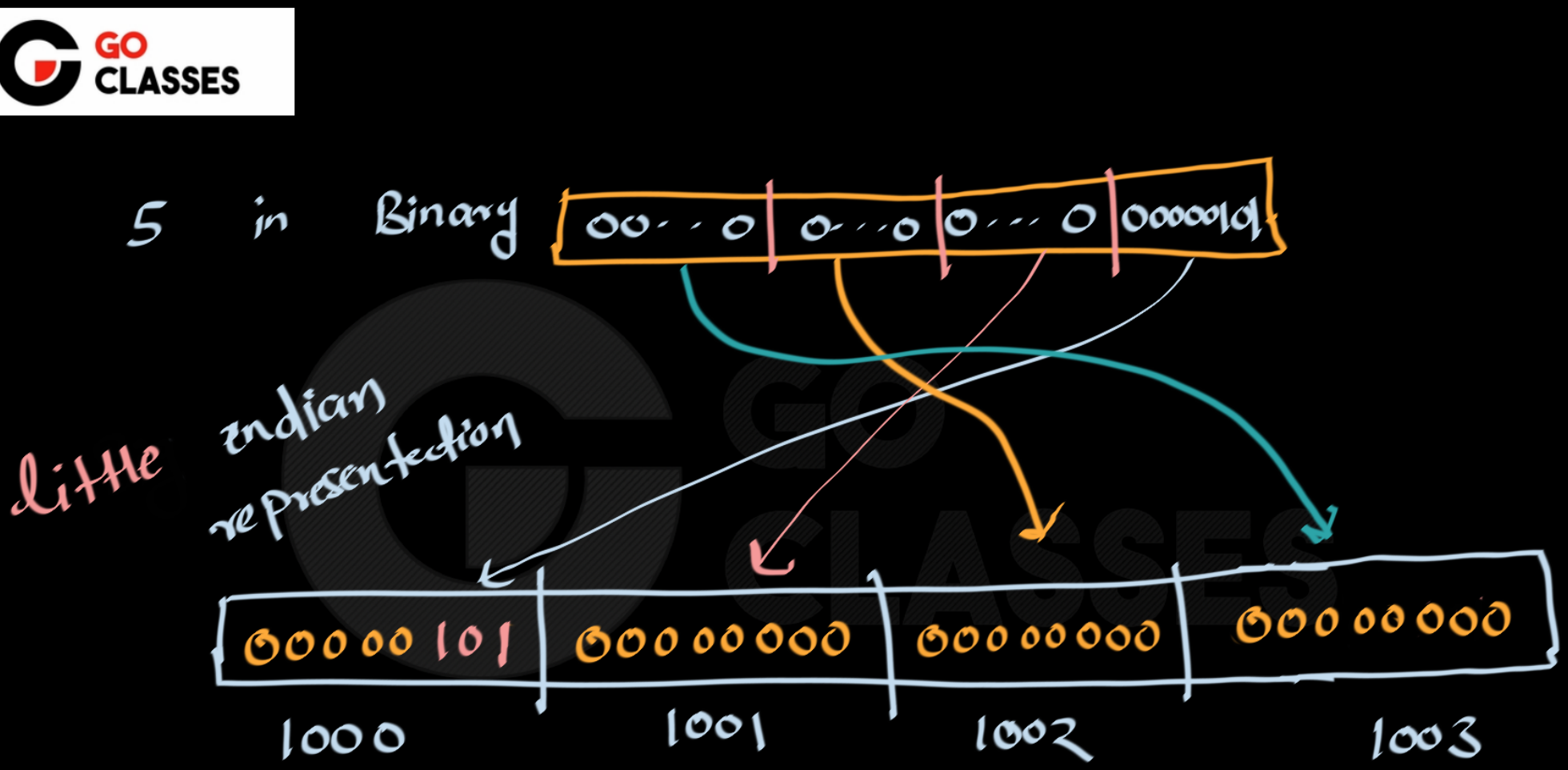
GO CLASSES

5 in Binary



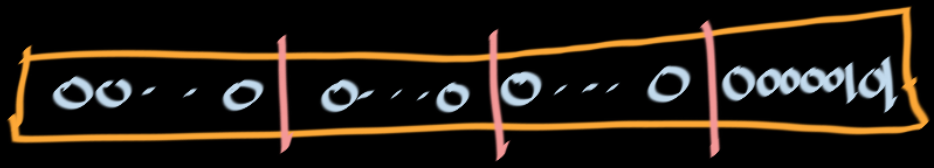
Big Endian Representation



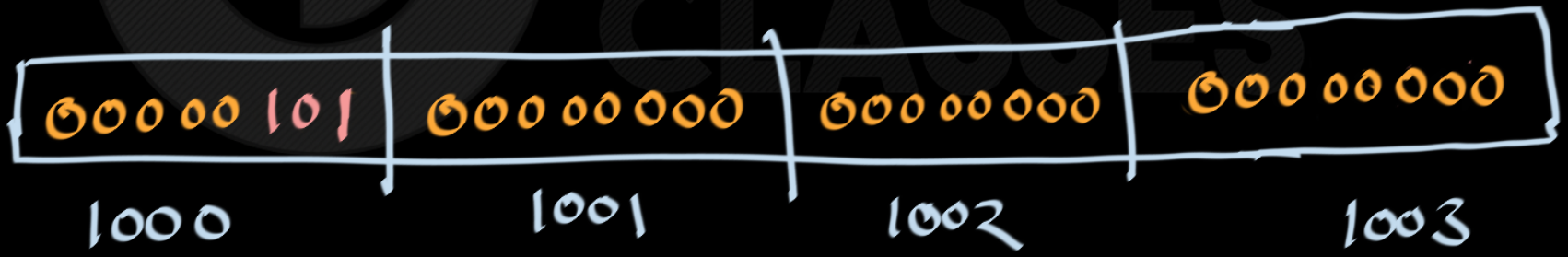

5 in Binary
00...0 | 0...0 | 0... 0 | 0000010
little endian representation
00000101 | 00000000 | 00000000 | 00000000
1000
1001
1002
1003

GO CLASSES

5 in Binary



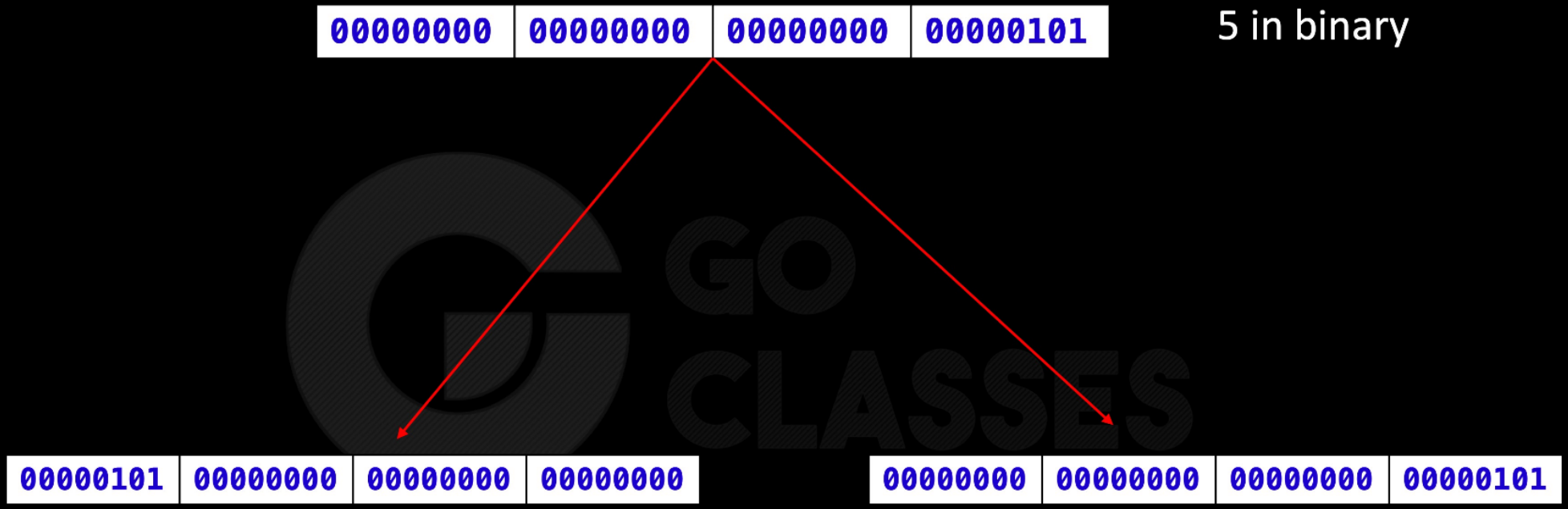
Little endian representation (more common)



Little endian vs Big endian

If data is multiple bytes then there are 2 ways to represent data – Little endian and Big Endian.

This is topic of computer architecture, here we will touch basics as required for C programming.


00000000 00000000 00000000 00000101
5 in binary
00000101 00000000 00000000 00000000
5 in Little endian
00000000 00000000 00000000 00000101
5 in Big endian

Endianness should not be reflected from user point of view.
Following code outputs same in both systems.

```
int i =5;
printf("%d",i ); //in both cases, output is always 5
```

Endianess is only applicable to single variables

4

Endianess is only applicable to single variables,
ie. only to each array element value, not to the order of elements in the array.

And 1-byte-variables can´t have their byte order reversed,
so no need to do anything with them.

Share Edit Follow

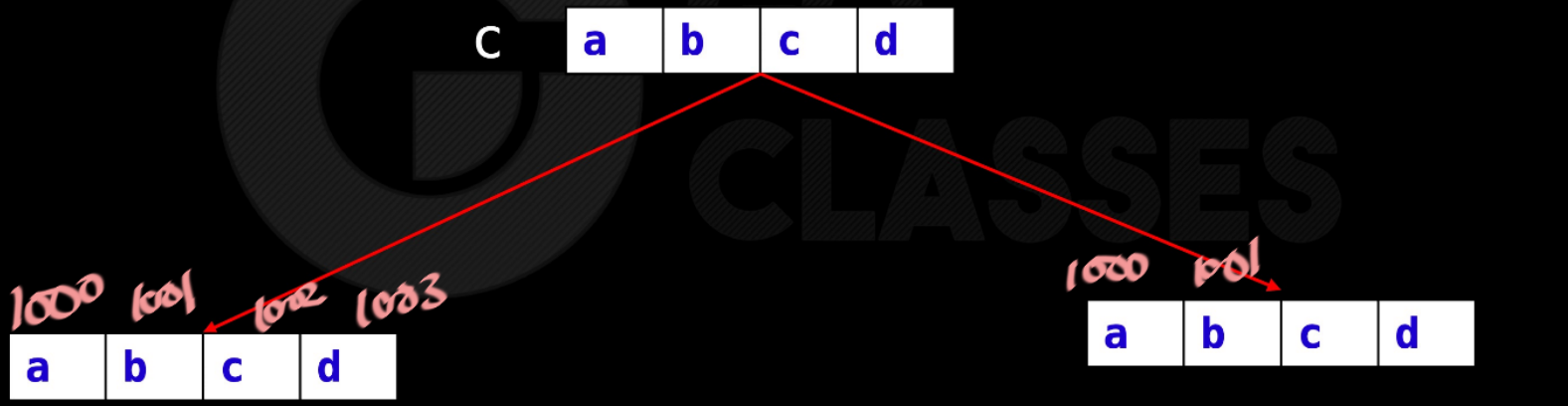
answered Feb 27, 2014 at 17:00

deviantfan
10.9k • 3 • 27 • 47

Add a comment

```
char c[] = {'a', 'b', 'c', 'd'};
```

Endianess can't change array order

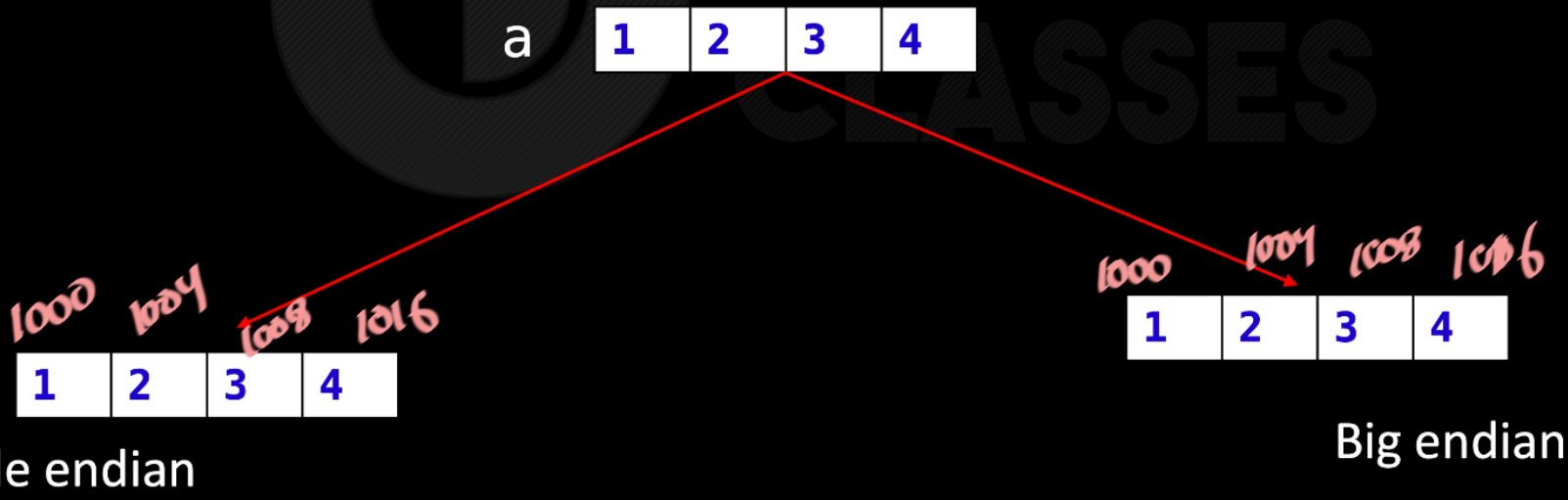


Little endian

Big endian

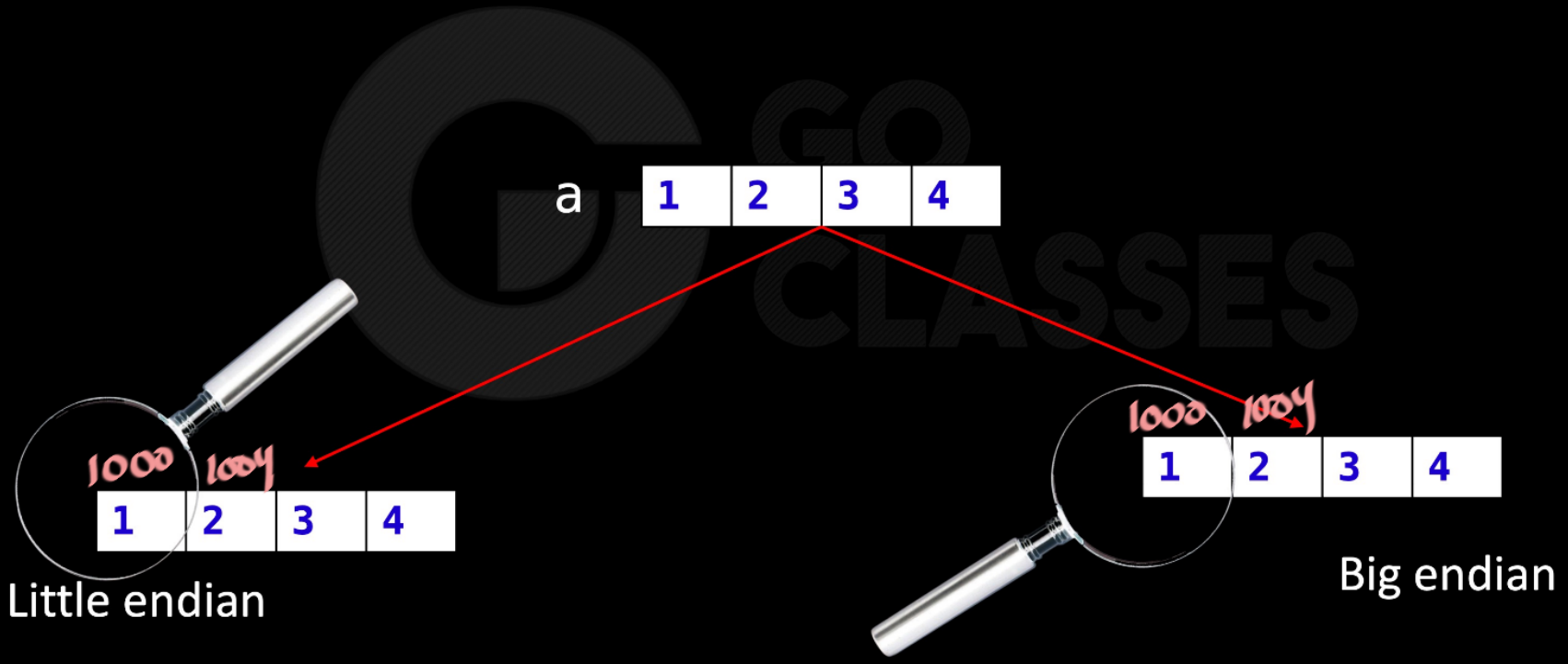
```
int a[] = {1, 2, 3, 4 }
```

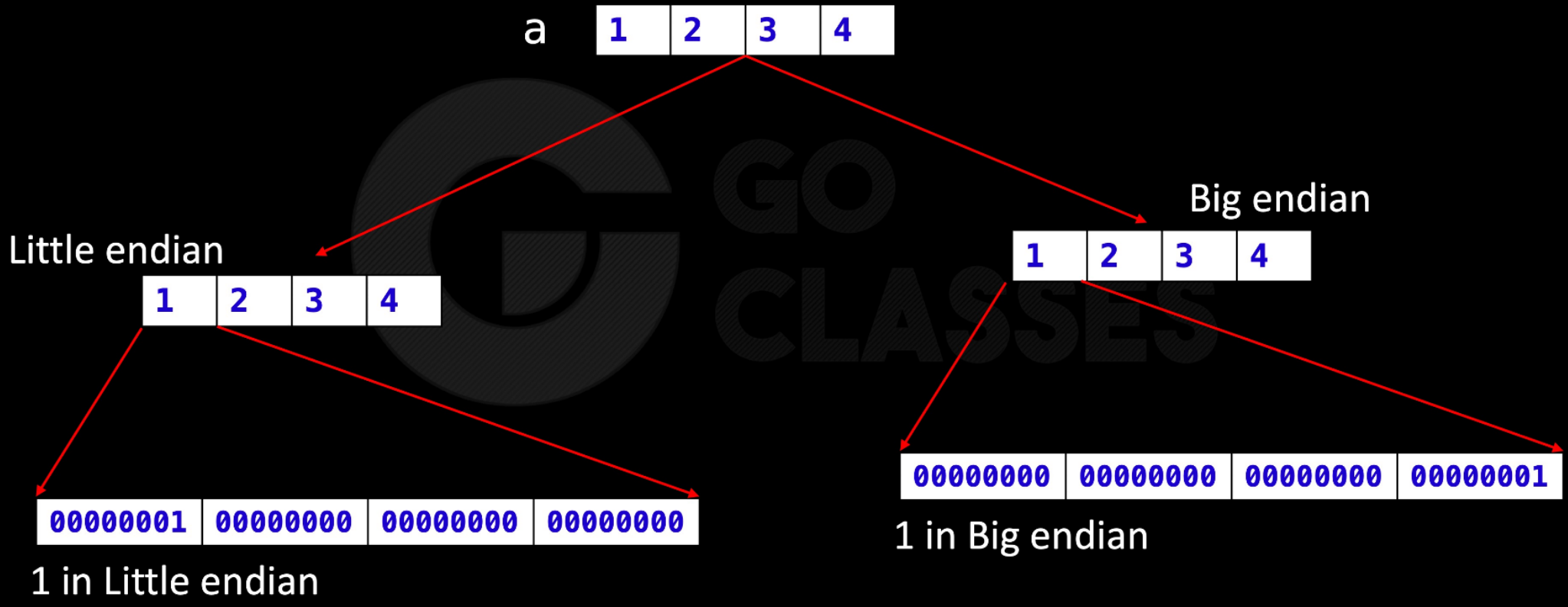
Endianess can't change array order

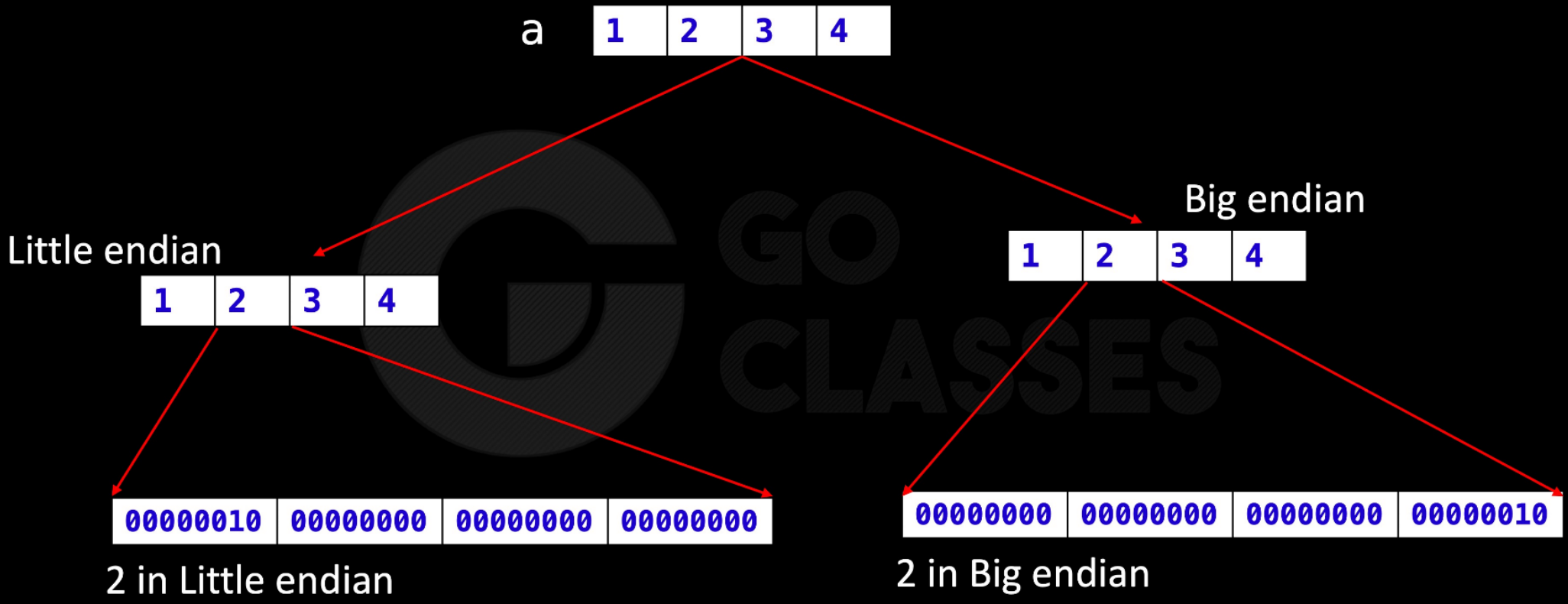


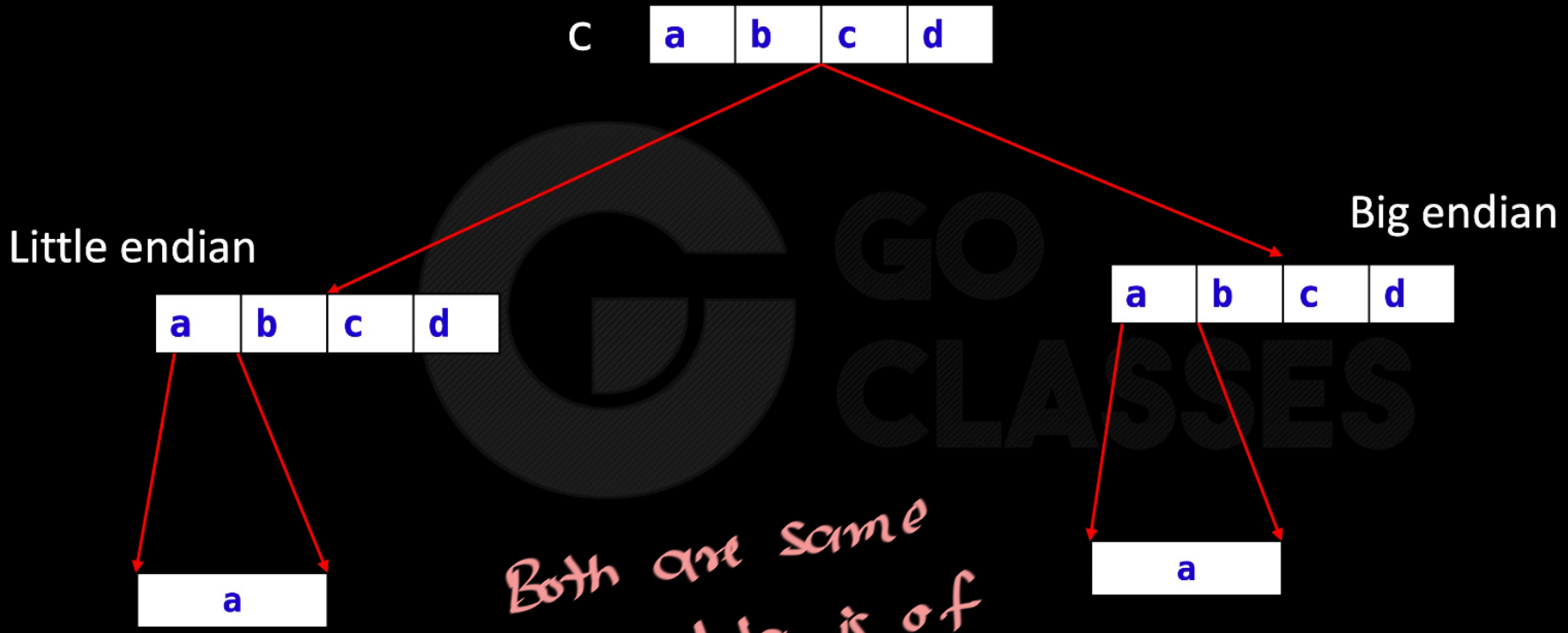
```
int a[] = {1, 2, 3, 4 }
```

Endianess can't change array order




a
1
2
3
4
Little endian
1
2
3
4
00000001
00000000
00000000
00000000
1 in Little endian
Big endian
1
2
3
4
00000000
00000000
00000000
00000001
1 in Big endian

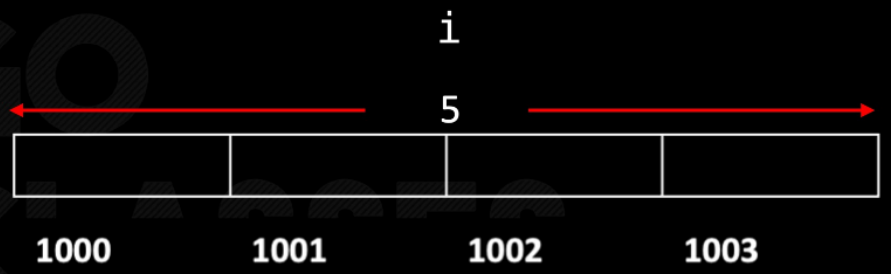

a
1
2
3
4
Little endian
1
2
3
4
Big endian
1
2
3
4
00000010
00000000
00000000
00000000
2 in Little endian
00000000
00000000
00000000
00000010
2 in Big endian



Both are same if data is of single byte

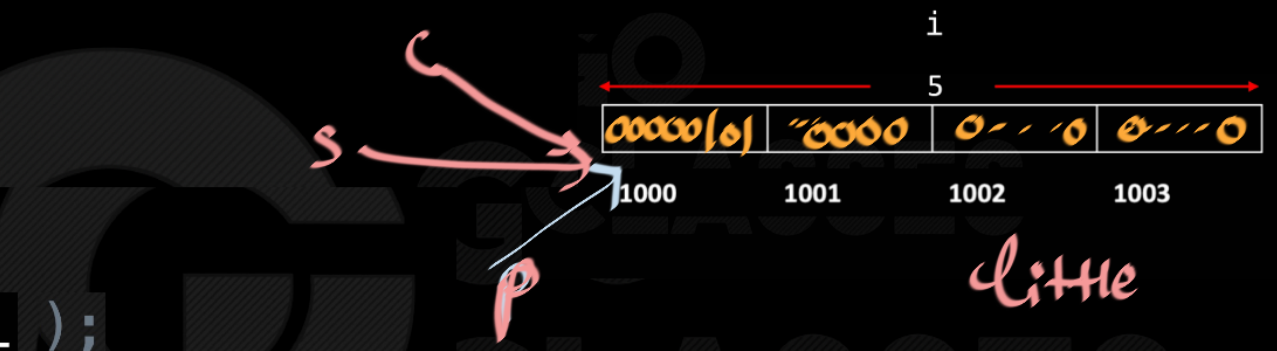
We can make any pointer to point to any location in memory

```
int i =5;


int *p;
short int *s;
char *c;
```



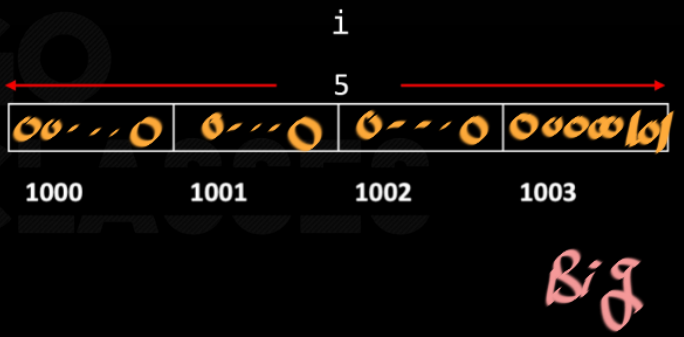
We can make any pointer to point to any location in memory

```
int i =5;
printf("%d",i );
```




```
int *p;
short int *s;
char *c;
```

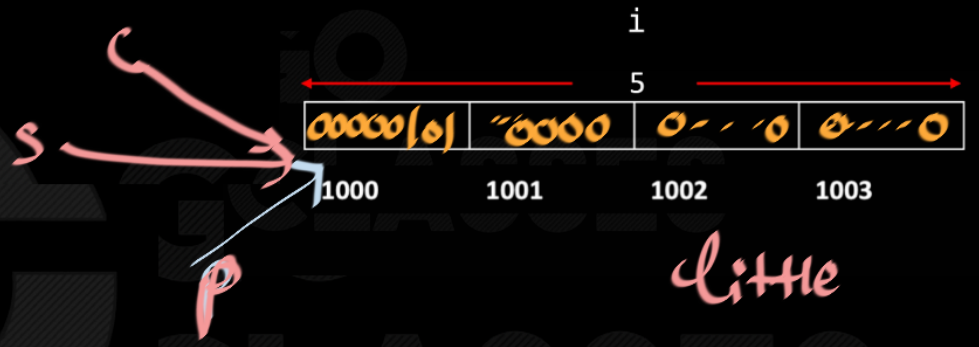
p = &i
s = &i
c = &i

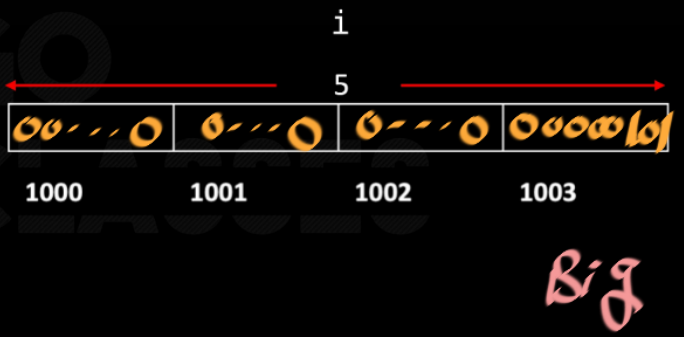


We can make any pointer to point to any location in memory

```
int i =5;
printf("%d",i );
```

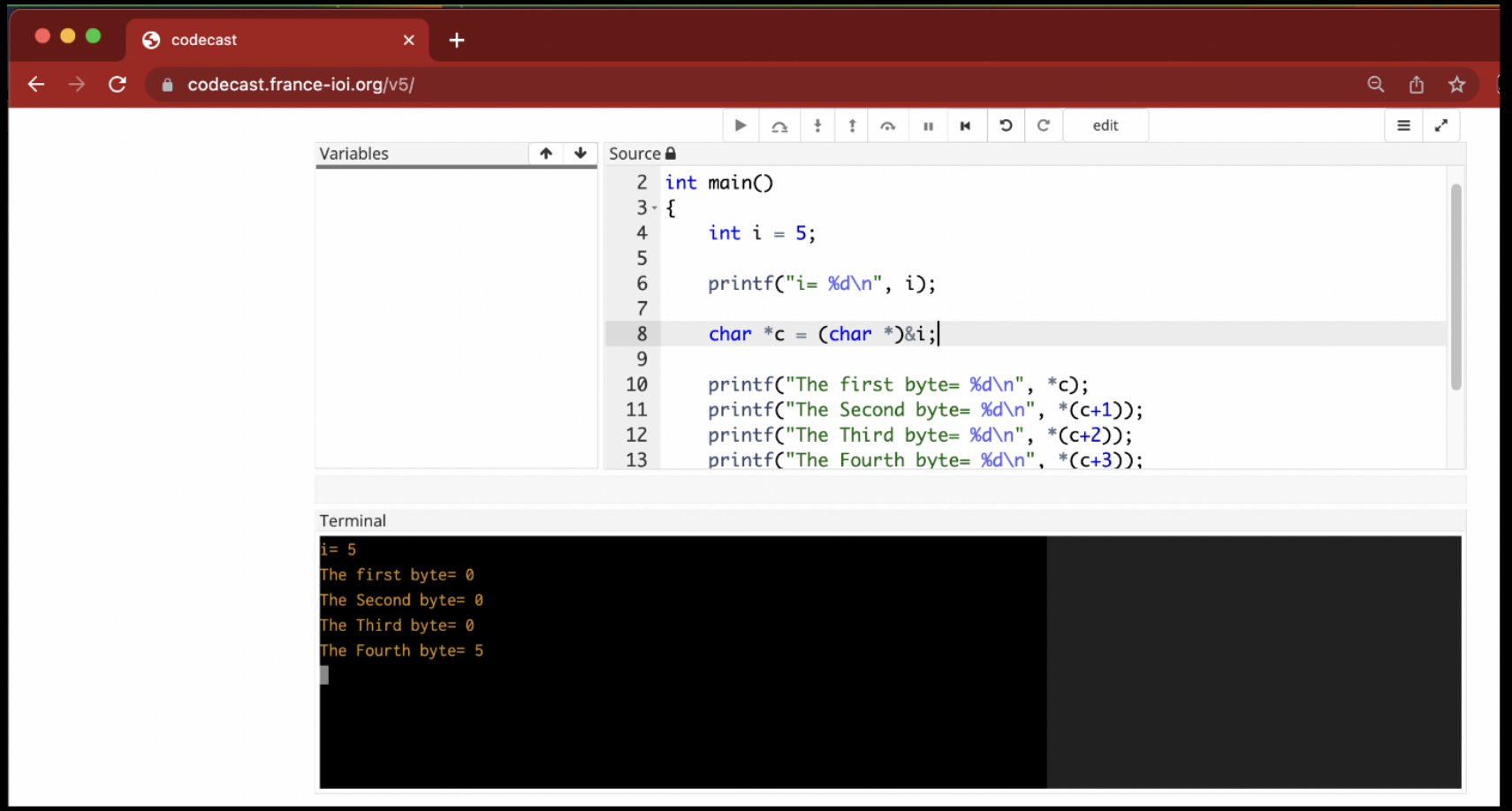
printf ("%d", *c);





GO CLASSES

Big endian



GO CLASSES

Little Endian

a.c

```c
#include<stdio.h>
int main()
{
    int i = 5;

    printf("i= %d\n", i);

    char *c = (char *)&i;

    printf("The first byte= %d\n", *c);
    printf("The Second byte= %d\n", *(c+1));
    printf("The Third byte= %d\n", *(c+2));
    printf("The Fourth byte= %d\n", *(c+3));


}
```

Desktop — -zsh — 80×24

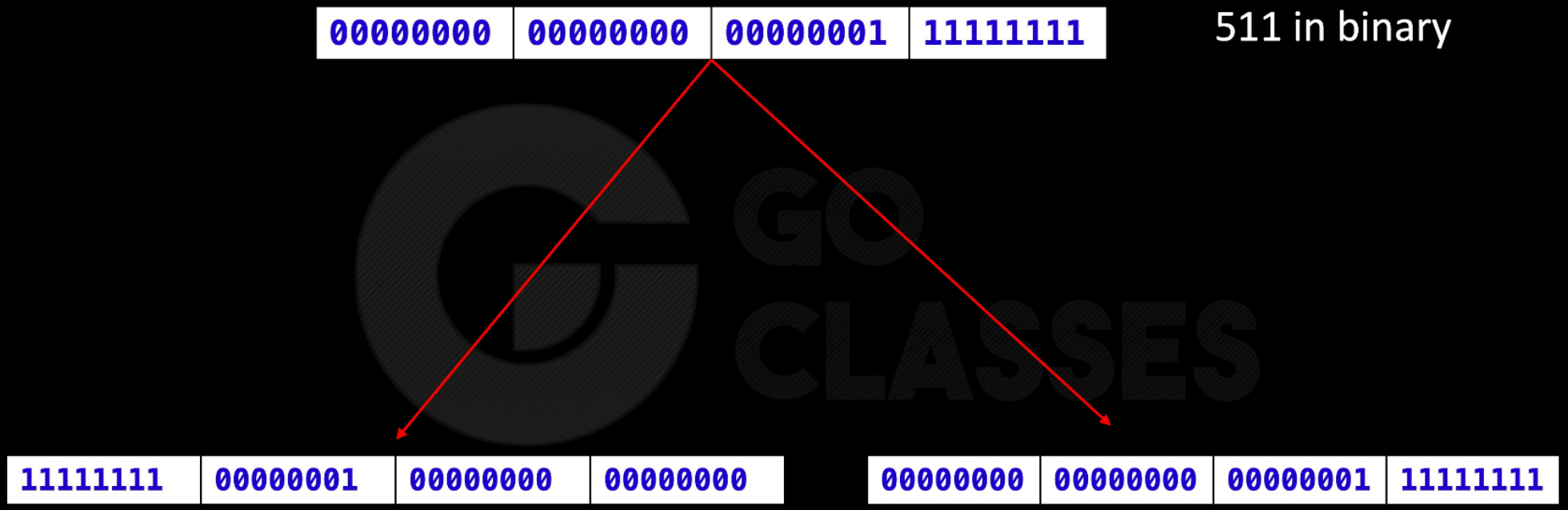
```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
i= 5
The first byte= 5
The Second byte= 0
The Third byte= 0
The Fourth byte= 0
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

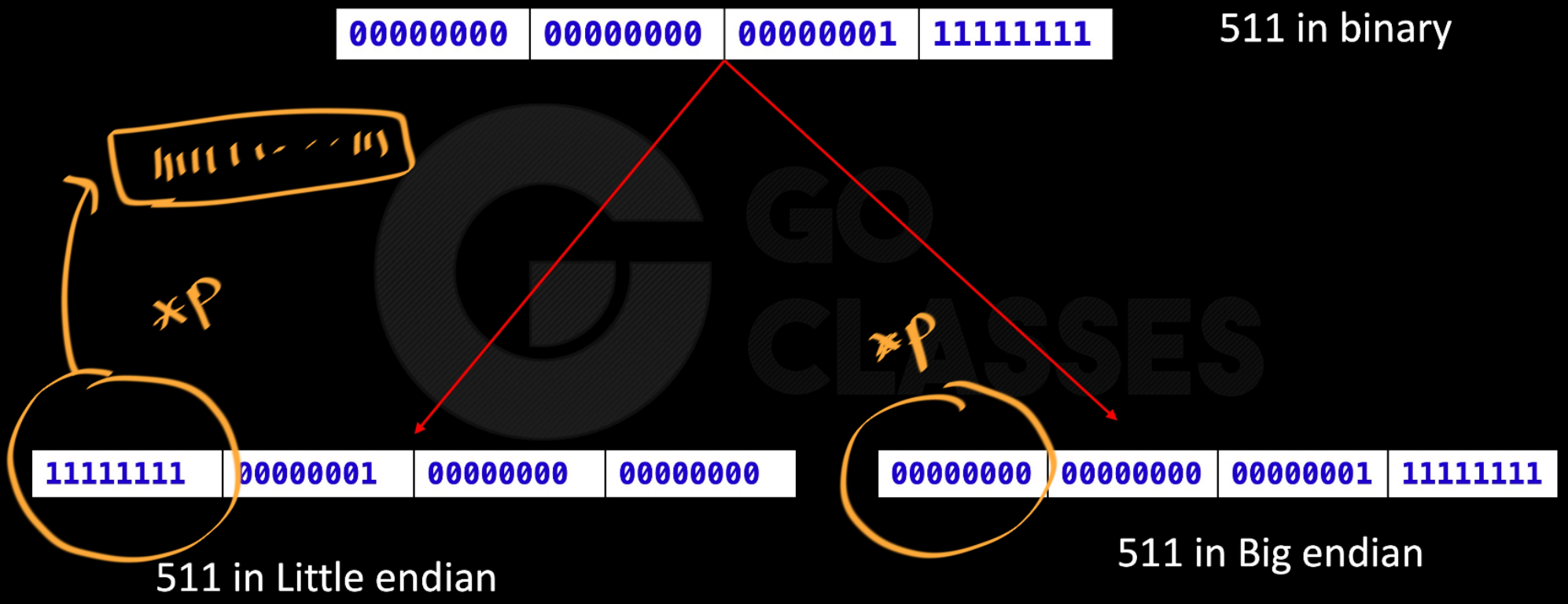
Question:

What will be the output of this program in both endianness ?

```c
#include <stdio.h>
int main(void){
    int i=511;
    signed char *p = (char *)&i;
    printf("%d", *p);
}
```

511 in binary -
00000000 00000000 00000001 11111111


00000000 00000000 00000001 11111111
511 in binary
11111111 00000001 00000000 00000000
511 in Little endian
00000000 00000000 00000001 11111111
511 in Big endian


00000000 00000000 00000001 11111111
511 in binary
*p
*p
11111111 00000001 00000000 00000000
00000000 00000000 00000001 11111111
511 in Little endian
511 in Big endian

GO CLASSES

Little endian

```c
#include<stdio.h>
int main()
{
    int i = 511;

    printf("i= %d\n", i);

    signed char *p = &i; //p is pointing to 8 bytes which are all ones
                         // in little endian system

    printf("The first byte= %d\n", *p);


}
```

```
Desktop — -zsh — 80×24
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c -w
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
i= 511
The first byte= -1
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

GO CLASSES

Little endian

```c
#include<stdio.h>
int main()
{
    int i = 511;

    printf("i= %d\n", i);

    unsigned char *p = &i; //p is pointing to 8 bytes which are all ones
                           // in little endian system

    printf("The first byte= %d\n", *p);


}
```

```
Desktop — -zsh — 80×24
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
i= 511
The first byte= 255
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

Question:

What will be the output of this program in both endianness ?

```c
#include<stdio.h>
int main()
{
    int i= 255;
    int  *p= &i;

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

}
```

255 in binary is

00000000	00000000	00000000	1111 1111

Solution

Big endian

00000000	00000000	00000000	1111 1111

p

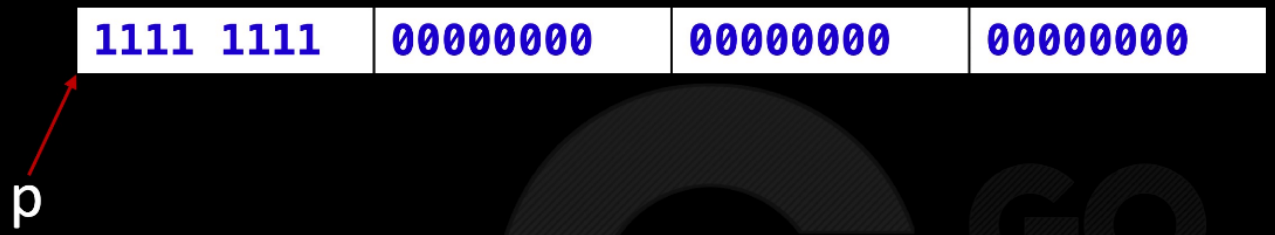
Little endian

1111 1111	00000000	00000000	00000000

p

Little endian stores in reverse way, also when we fetch some value, it first **reverse** and convert to decimal.

Little endian



Getting first 4 bytes using *p :-

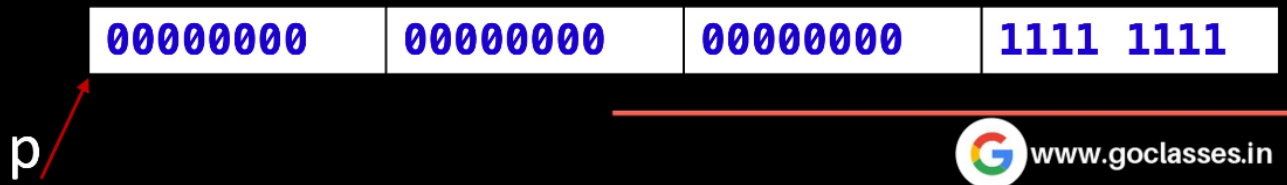
00000000	00000000	00000000	1111 1111

(notice we have revered the pattern)

This is 255 in signed or unsigned both

255 is answer

In case of Big endian, *p is 255 only

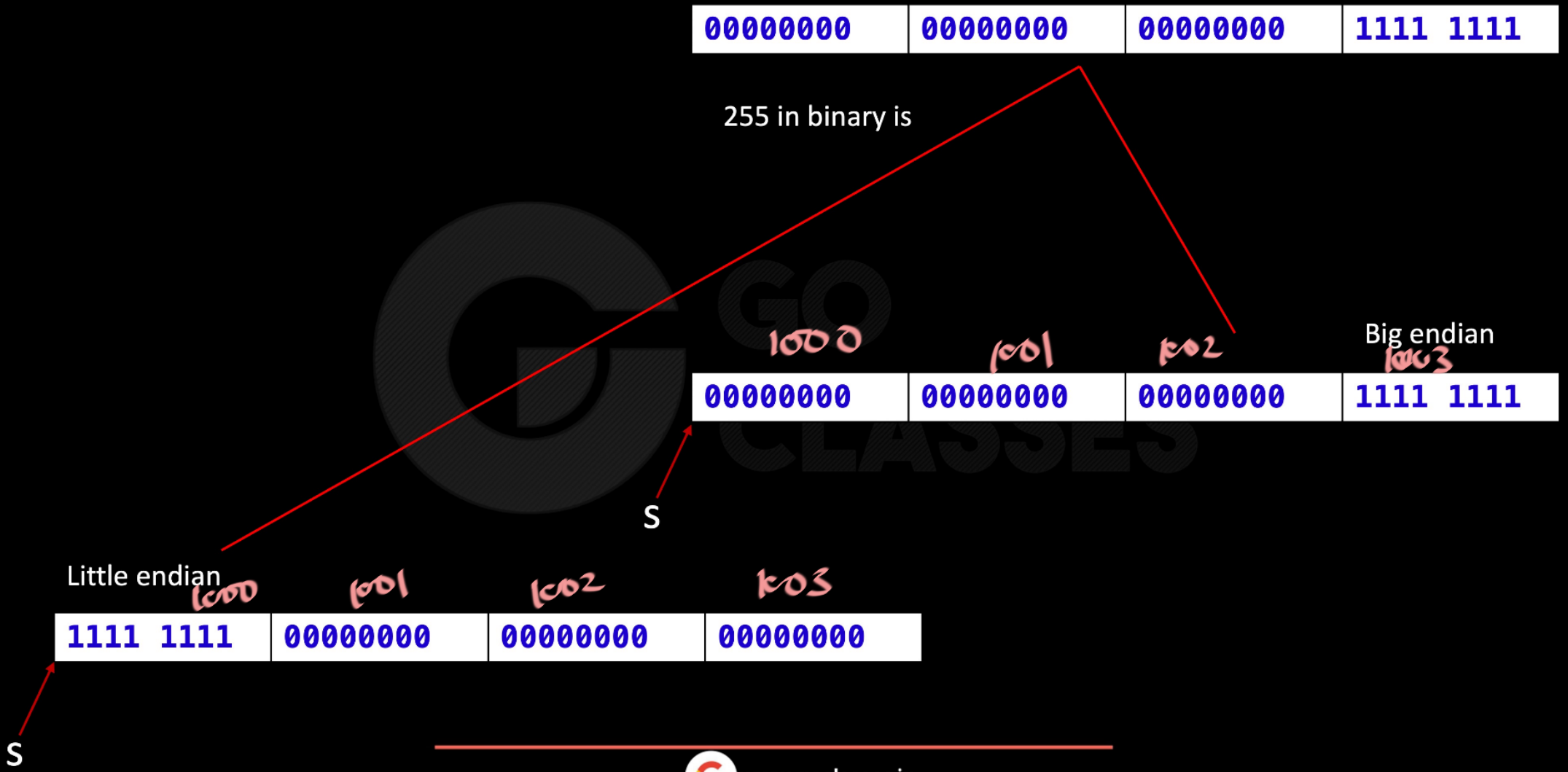


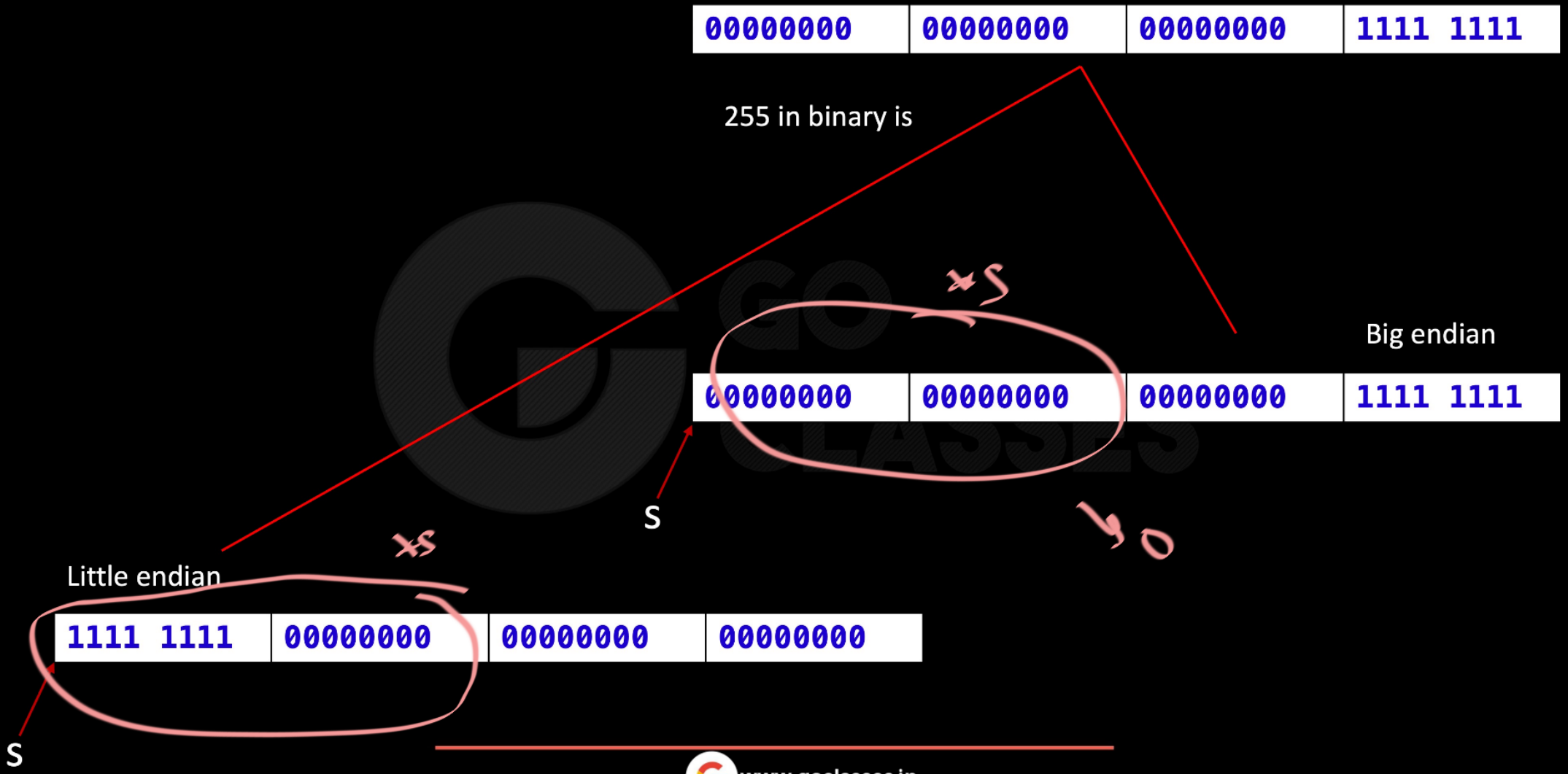
255 is answer

Question:

What will be the output of this program in both endianness ?

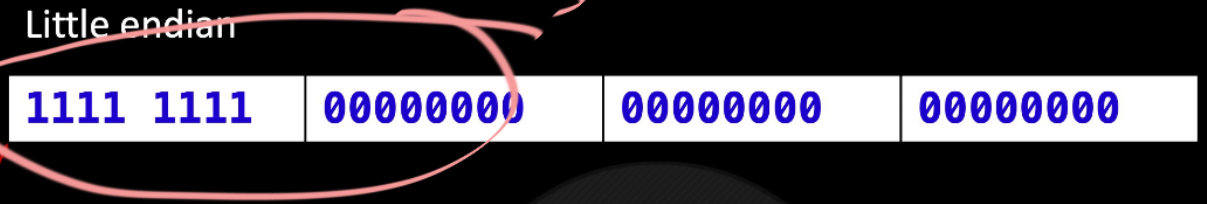
```c
#include<stdio.h>
int main()
{
    int i= 255;
    short int  *s= &i;

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

}
```


00000000 00000000 00000000 1111 1111
255 in binary is
1000 1001 1002 1003
Big endian
00000000 00000000 00000000 1111 1111
S
Little endian
1000 1001 1002 1003
1111 1111 00000000 00000000 00000000
S


00000000 00000000 00000000 1111 1111
255 in binary is
Big endian
00000000 00000000 00000000 1111 1111
S
Little endian
1111 1111 00000000 00000000 00000000
S

Little endian stores in reverse way, also when we fetch some value, it first **reverse** and convert to decimal.

Little endian

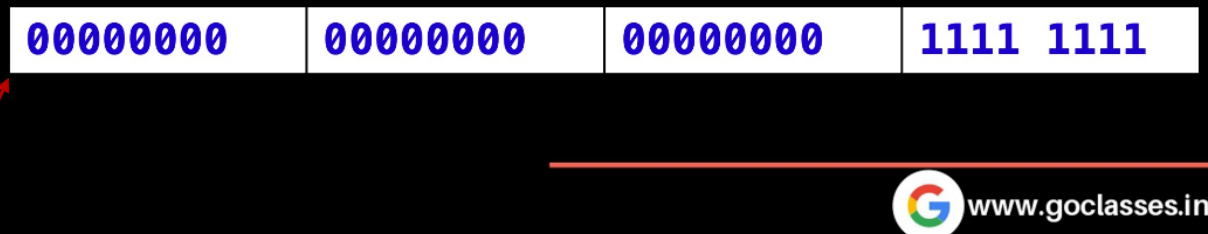


Getting first 2 bytes using *s :-

This is 255 in signed or unsigned both

255 is answer

In case of Big endian, *s is just **zero**



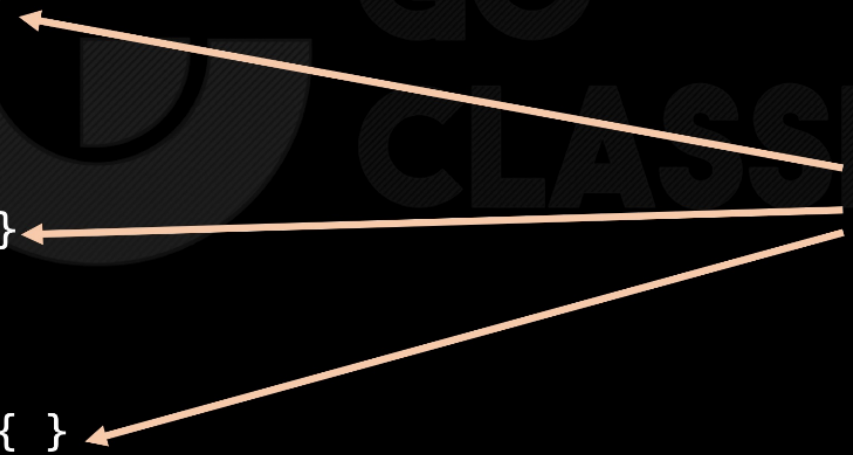
0 is answer

Passing 1D array to function

Passing 1D array to function

Copies the base address of array to function parameter

```
void funcl(int *a) { }

void func2(int a[]) { }

void func3(int a[10]) { }
```



GO CLASSES

```
main( )
{
    int a[5] = {1,2,3,4,5};
    fun ( )
}
```

```
main( )
{
  int i ;
  fun( i )
}
void fun( int x)
  {
    pf (x+1)
  }
```

```
main( )
{
  int a[5] = {1,2,3,4,5};
  fun( a )
}
void fun( )
  {
  }
```

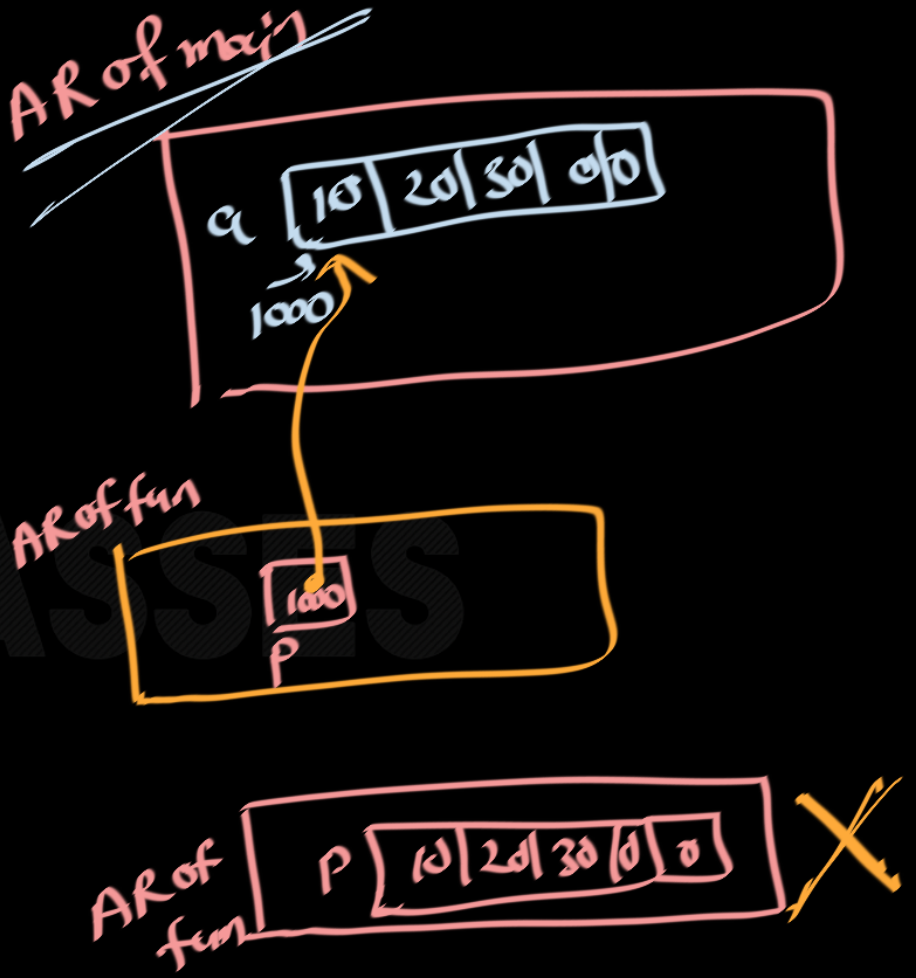
GO CLASSES

```
main( )
{
  int a[5] = {1, 2, 3, 4, 5};
  fun(a)        <- 1000
}
void fun( int *p )
{
}
```

int *p = a ;

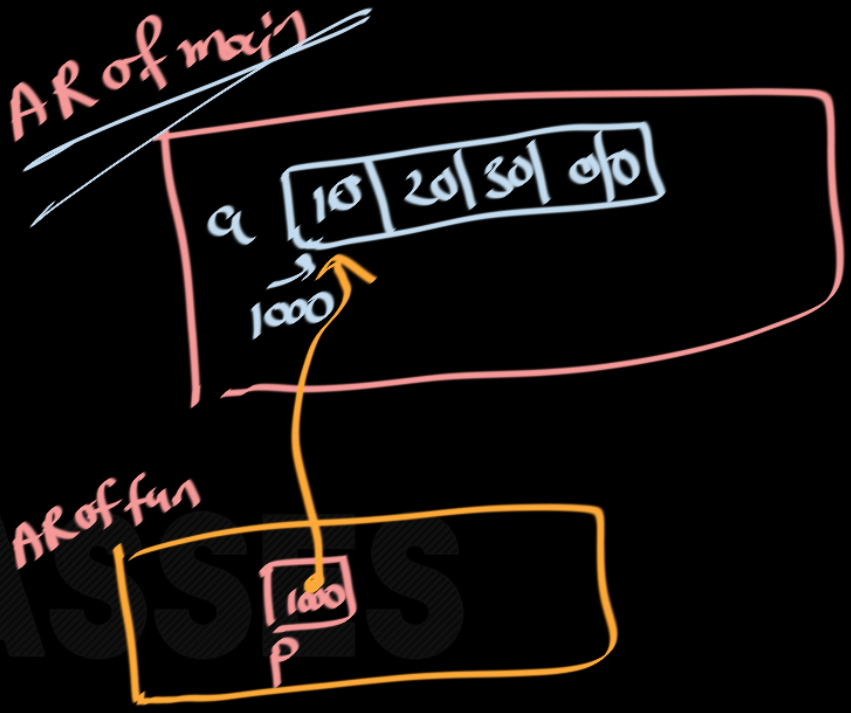
GO CLASSES

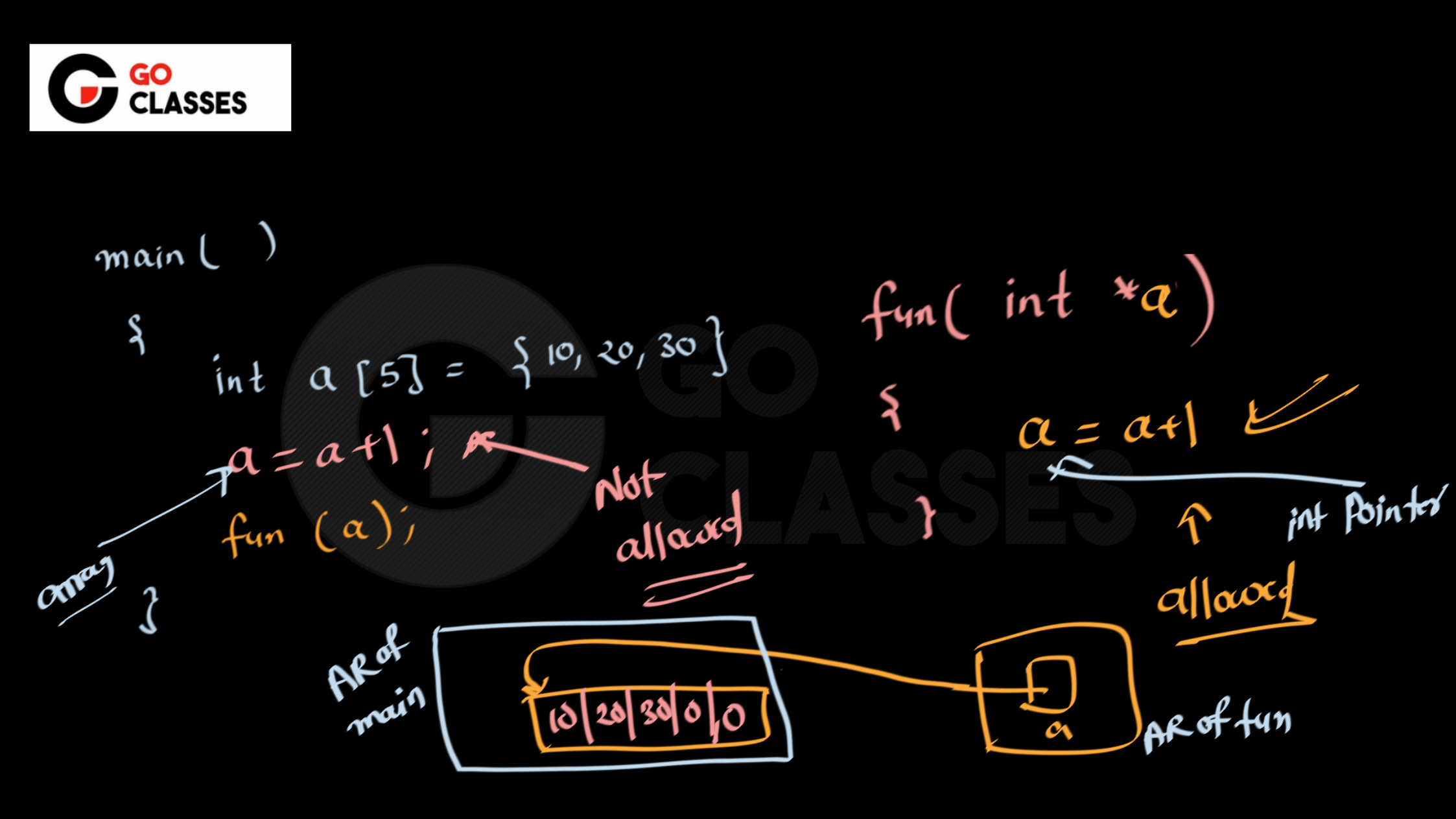
```
main( )
{
    int a[5] = {10, 20, 30}
    fun (a);
}
fun( int *p)
{
}
```



GO CLASSES

```
main( )
{
    int a[5] = { 10, 20, 30 }
    fun (a);
      ↑ 1000
}
fun( int *p)
{
}
```




main ( )
{
int a [5] = { 10, 20, 30 }
a = a+1 ;
fun (a);
}
array
Not allowed
AR of main
10 20 30 0 0
fun (int *a)
{
a = a+1
}
int Pointer
allowed
a
AR of fun


```
main( )
{
    int a[] = {1, 2};
    fun(a);
}
```

```
void fun( )
{
}
```

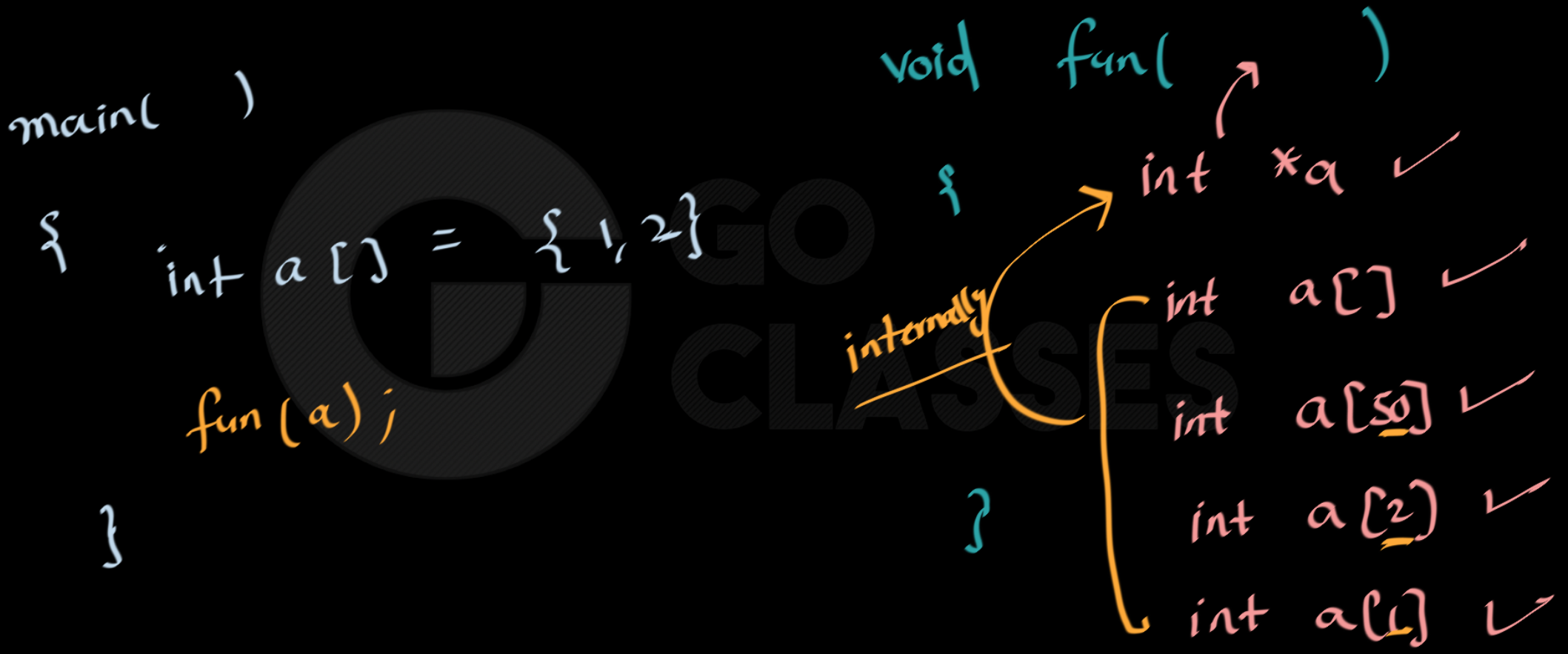
int *a

int a[]

internally

Beginner

int a[50]


main( )
{
int a[] = {1, 2}
fun(a);
}
void fun( )
{
}
internally
int *a
int a[]
int a[50]
int a[2]
int a[1]

GO CLASSES

```
main( )
{
    int arr = {10, 0, 3, 5}
    fun(arr);
    Pf(arr[0], arr[1] - -);
}
```

arr = arr + 2 ✗

```
Void fun(int a[])
{
    a = a + 1 ✓ allowed
}
```


```
main( )
{
    int arr = { 10, 0, 3, 5}
    fun(arr);
    Pf ( arr[0] , arr[1] , - , - )
}
```

11 3 3 5

```
Void fun( int a[ ])
{
    *a = *a+1
    a++;
    *a = *a + *(a+1)
}
```

0 3

Passing 1D array to function (contd..)

- What really gets passed is a pointer to the array's first element
- We can use parameter name on the left side of assignment inside the called function.

Strings in C programming

GO CLASSES

int a;

char c;

String s;

do we have something like this? ⇒ No.

GO CLASSES

```
char c[10];
```

C

we have only character array in the C programming but we don't have strings.

You can treat the character array as a string in C

GO CLASSES

```
char c[10] = "Hello";
```

- c[0] is 'H'
- c[1] is 'e'

```
printf("%s", c);
```

c → 1000

↳ it will print everything from the address 1000 till it gets null char.

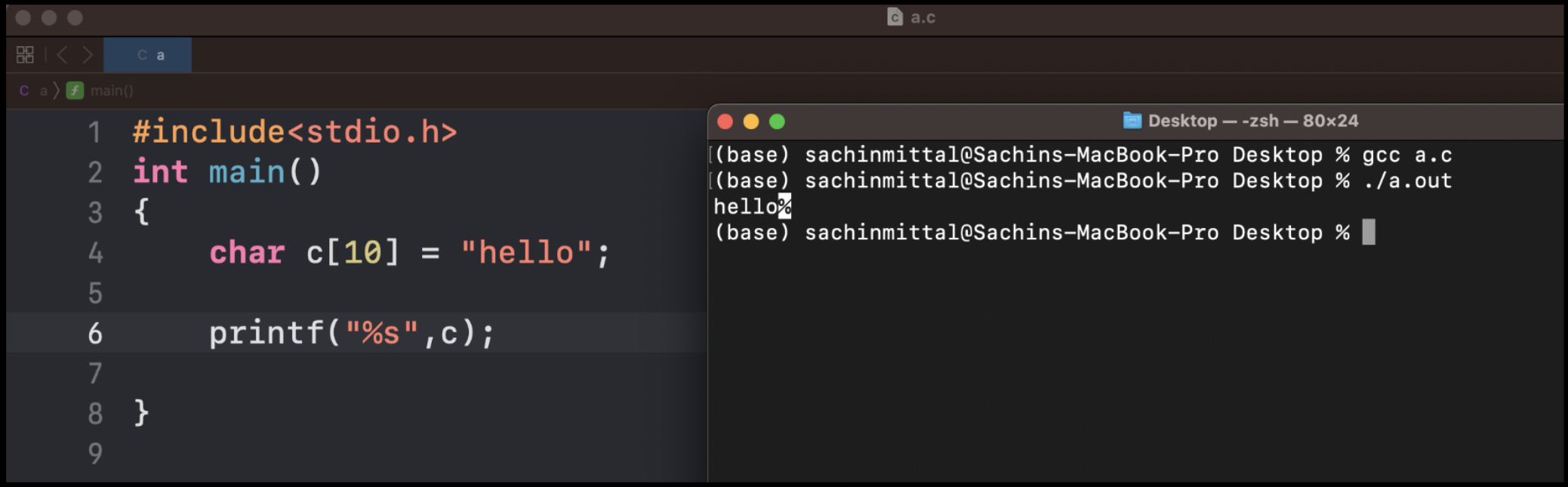
GO CLASSES

```
Char c[10] = "Hello";
```

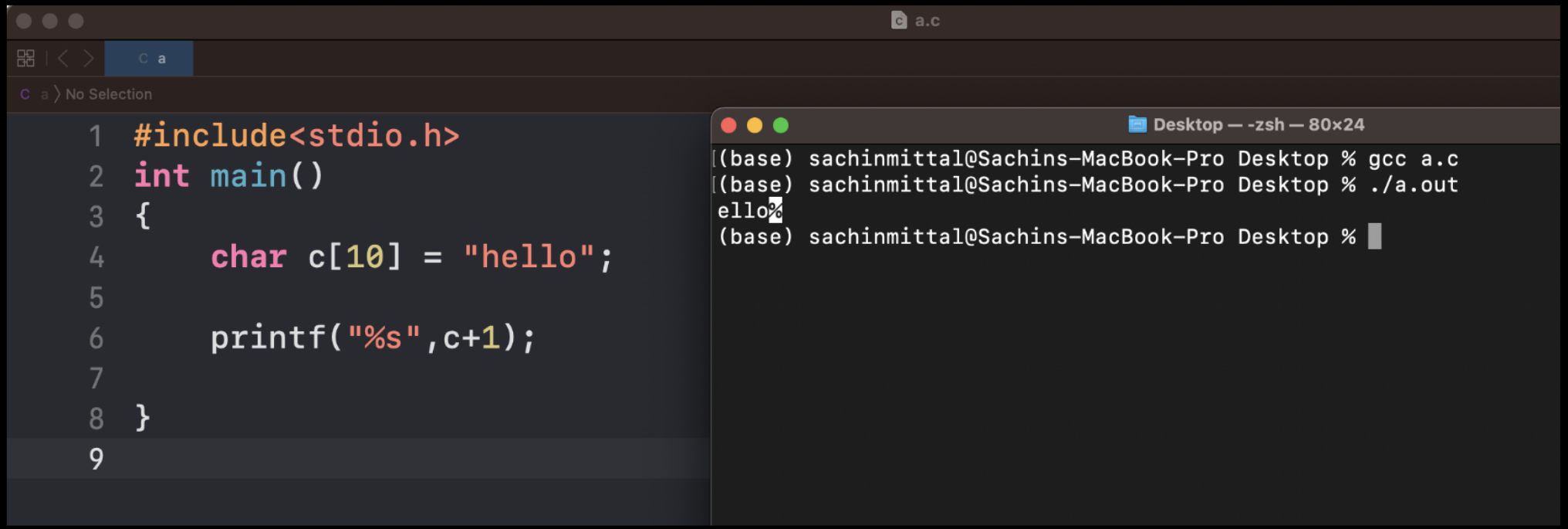
1000


```
printf("%s", c);
```

→ Hello


a.c
a
a
main()
1 #include<stdio.h>
2 int main()
3 {
4 char c[10] = "hello";
5
6 printf("%s",c);
7
8 }
9
Desktop — -zsh — 80×24
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
hello%
(base) sachinmittal@Sachins-MacBook-Pro Desktop %

GO CLASSES


a.c
c a
c a 〉No Selection
1 #include<stdio.h>
2 int main()
3 {
4 char c[10] = "hello";
5
6 printf("%s",c+1);
7
8 }
9
Desktop — -zsh — 80×24
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
ello%
(base) sachinmittal@Sachins-MacBook-Pro Desktop %

String is not an explicit type, instead array of character can be treated as string if it ends with null character

GO CLASSES

```
main(  )
{
    char  c[] = "GoClasses";
    int  len = strlen(c);
}
```

```
int  strlen(  )
{
```

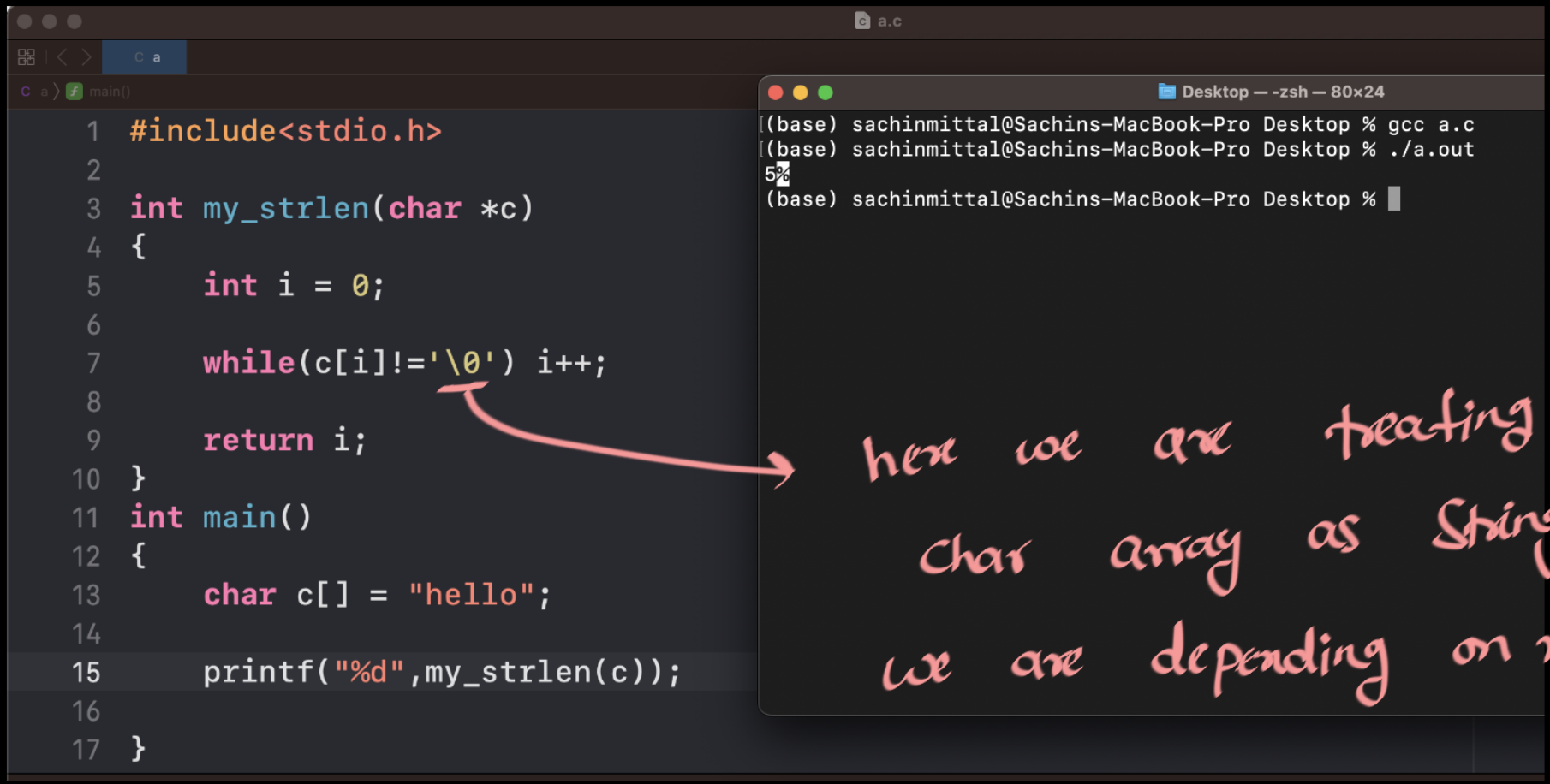
```
char *c
char  c[]
char c[500]
```

```
}
```

GO CLASSES

```
main( )
{
    char c[] = "GoClasses";
    int len = Strlen(c);
}
```

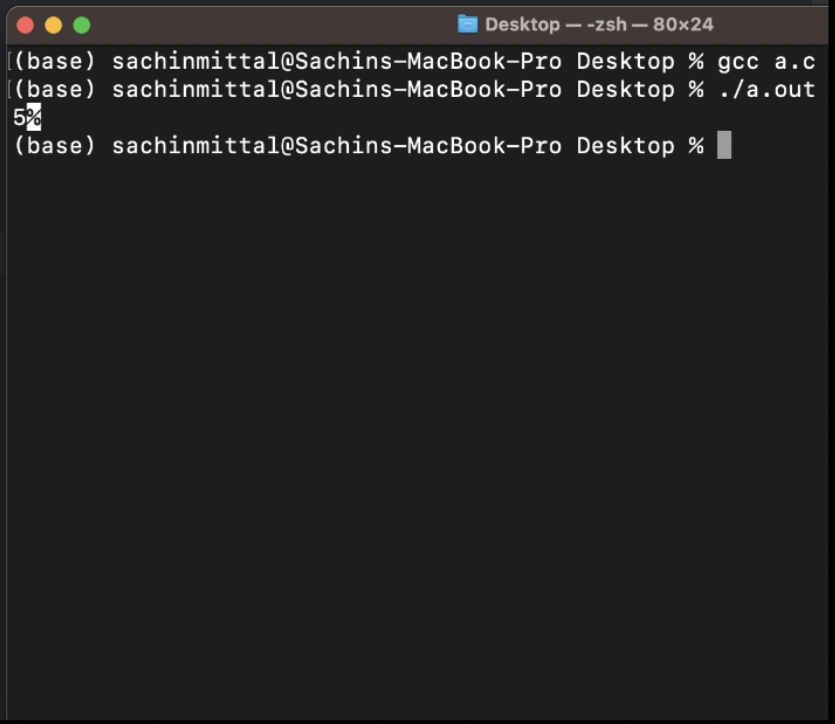
```
int Strlen( char *c )
{
    int i = 0;
    while (c[i] != '\0')
        i++;
    return i;
}
```


a.c
a
main()
1 #include<stdio.h>
2
3 int my_strlen(char *c)
4 {
5 int i = 0;
6
7 while(c[i]!='\0') i++;
8
9 return i;
10 }
11 int main()
12 {
13 char c[] = "hello";
14
15 printf("%d",my_strlen(c));
16
17 }
Desktop — -zsh — 80×24
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
5%
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
here we are treating
char array as String bcoz
we are depending on null char.

GO CLASSES

another version of mystrlen function.

```c
int my_strlen(char *c)
{
    int i = 0;

    while(*c !='\0'){
        c++;
        i++;
    }

    return i;
}
int main()
{
    char c[] = "hello";

    printf("%d",my_strlen(c));

}
```



String litterals in C programming

```
main()
{
  char *c = "Hello";
}
```

↳ String litteral

GO CLASSES

char c[] = "Hello" c[0] = 'g'; ✓

char *t = "Hello" t[0] = 'g'; ✗

↳ this is constant and we can not modify this (ROM in Static area)

GO CLASSES

```c
#include<stdio.h>

int main()
{
    char c[] = "hello";
    char *t = "hello";

    c[0] = 'g';
    //t[0] = 'g'; //NOT allowed to modify string litterals

}
```

GO CLASSES

a.c

```c
#include<stdio.h>


int main()
{
    char c[10] = "hello";

    for(int i = 0; i<5;i++){
        printf("%c  ",c[i]);
    }

}

```

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
h  e  l  l  o  %
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

Desktop — -zsh — 80×24

here we are treating each element as an individual element.

try to write a function in C

s.t it compares two strings

s_1 "abc"

s_2 "abd"

$s_1 > s_2$ false

$s_2 > s_1$ true

$s_1 > s_2$ return 1

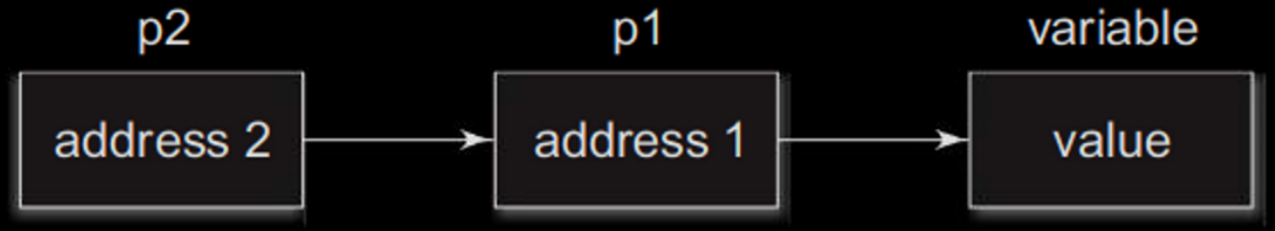
$s_2 > s_1$ return -1

$s_1 = s_2$ return 0

"abc" "ab" return 1;

Double Pointer

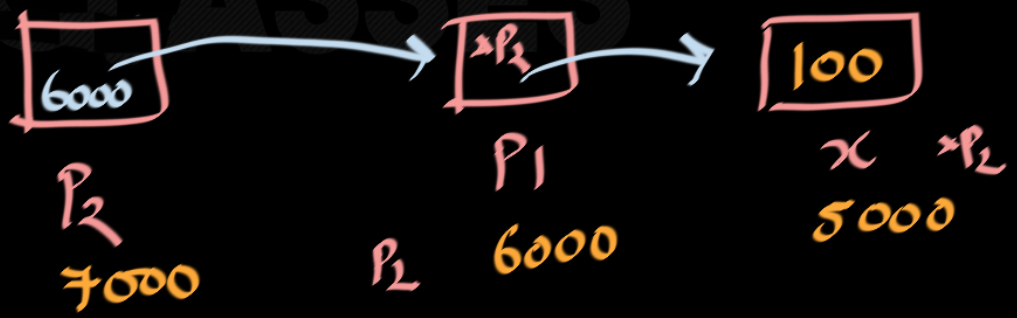
Pointers to Pointers to...



p2 is a pointer to a pointer

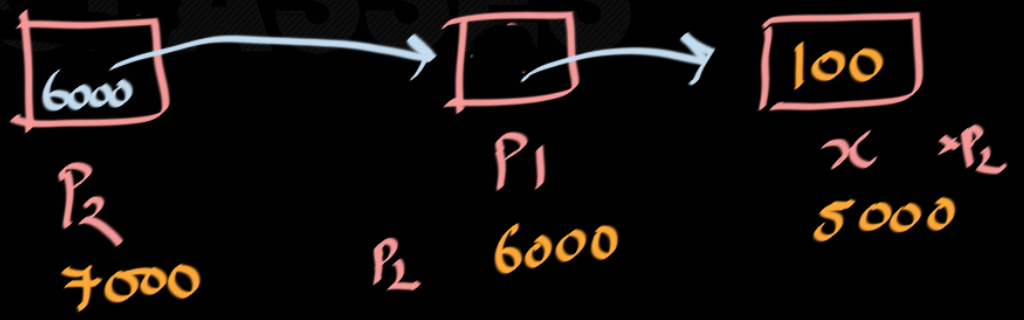
```
int x, *p1, **p2;
x = 100;
p1 = &x; /* address of x */
p2 = &p1 /* address of p1 */
printf ("%d", **p2);
```

→ 100



```c
int x, *p1, **p2;
x = 100;
p1 = &x; /* address of x */
p2 = &p1 /* address of p1 */
printf ("%d", **p2);
```

→ 100



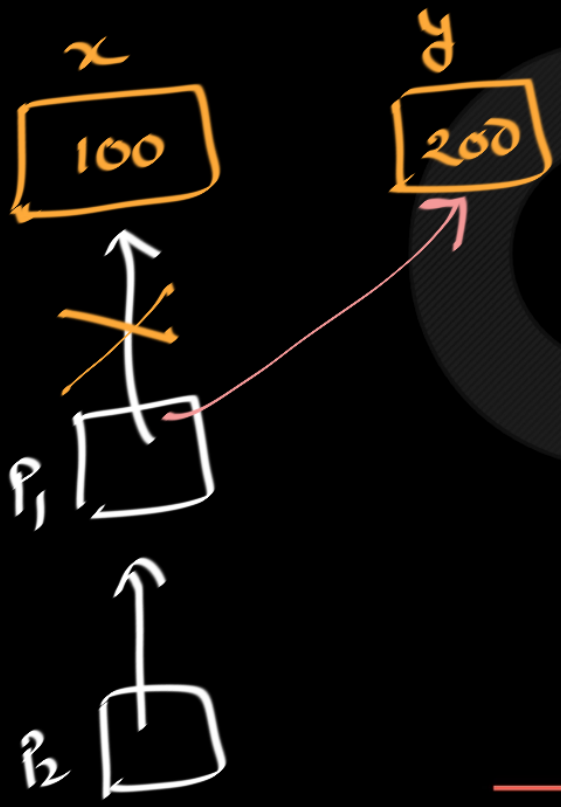
Different ways to access 100.

x

**P2

*P1

Question 1




```c
main() {

    int x, y, *p1, **p2;
    x = 100;
    y = 200;

    p1 = &x;
    p2 = &p1;

    printf("%d ", **p2);

    *p2 = &y;

    printf("%d ", **p2);

}
```

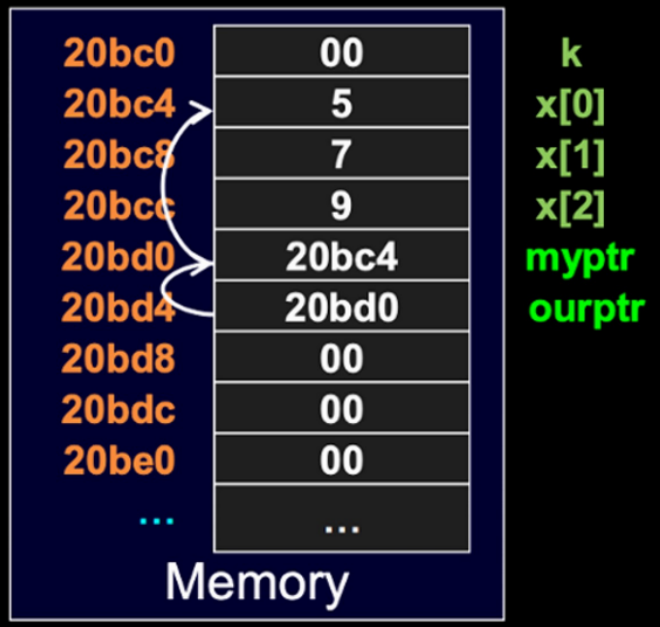
100

(same as saying $p_1 = \&y$)

200

Question 2

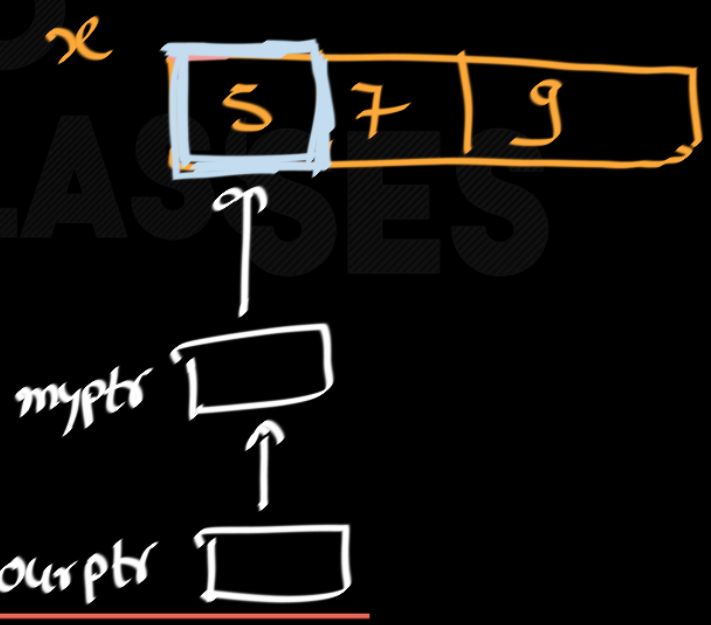
- Example
  - `int k, x[3] = {5, 7, 9};`
  - `int *myptr, **ourptr;`
  - `myptr = x;`
  - `ourptr = &myptr;`
  - `k = *myptr;` `// k=?`
  - `k = (**ourptr) + 1;` `// k=?`
  - `k = *(*ourptr + 1);` `// k=?`



Solution :-

- Example
  - `int k, x[3] = {5, 7, 9};`
  - `int *myptr, **ourptr;`
  - `myptr = x;`
  - `ourptr = &myptr;`
  - `k = *myptr;` // k=? 5
  - `k = (**ourptr) + 1;` // k=? 6
  - `k = *(*ourptr + 1);` // k=? 7

pf(*myptr) → 5

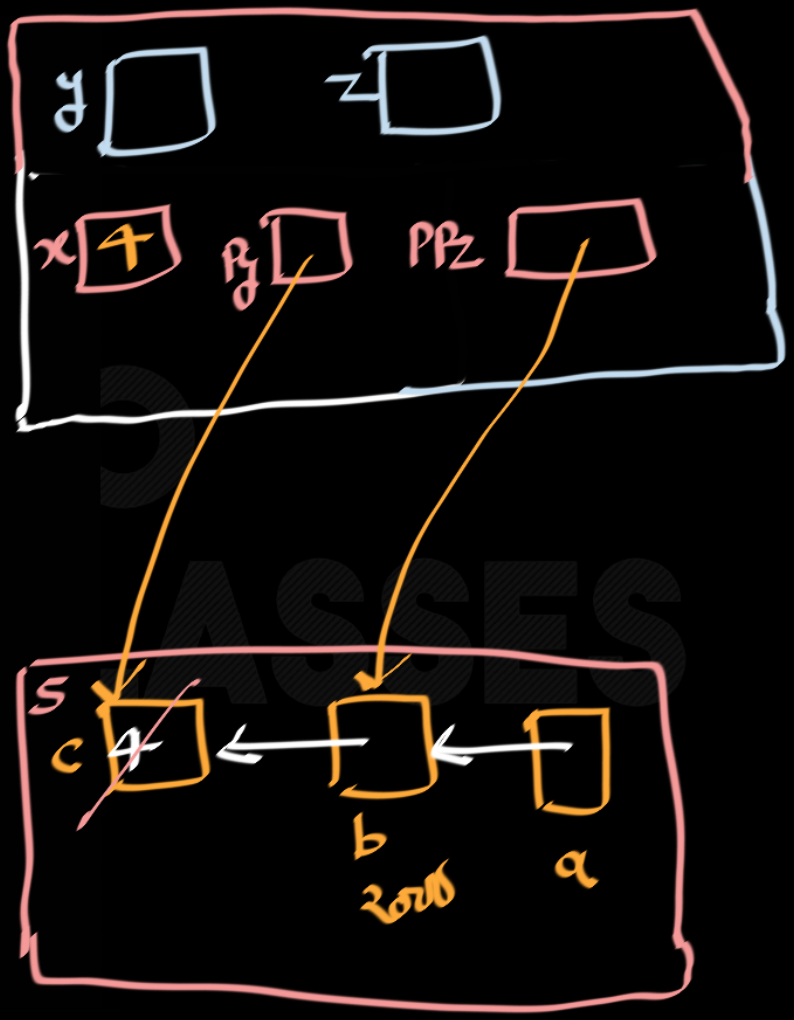


Question 3:

GATE 2008

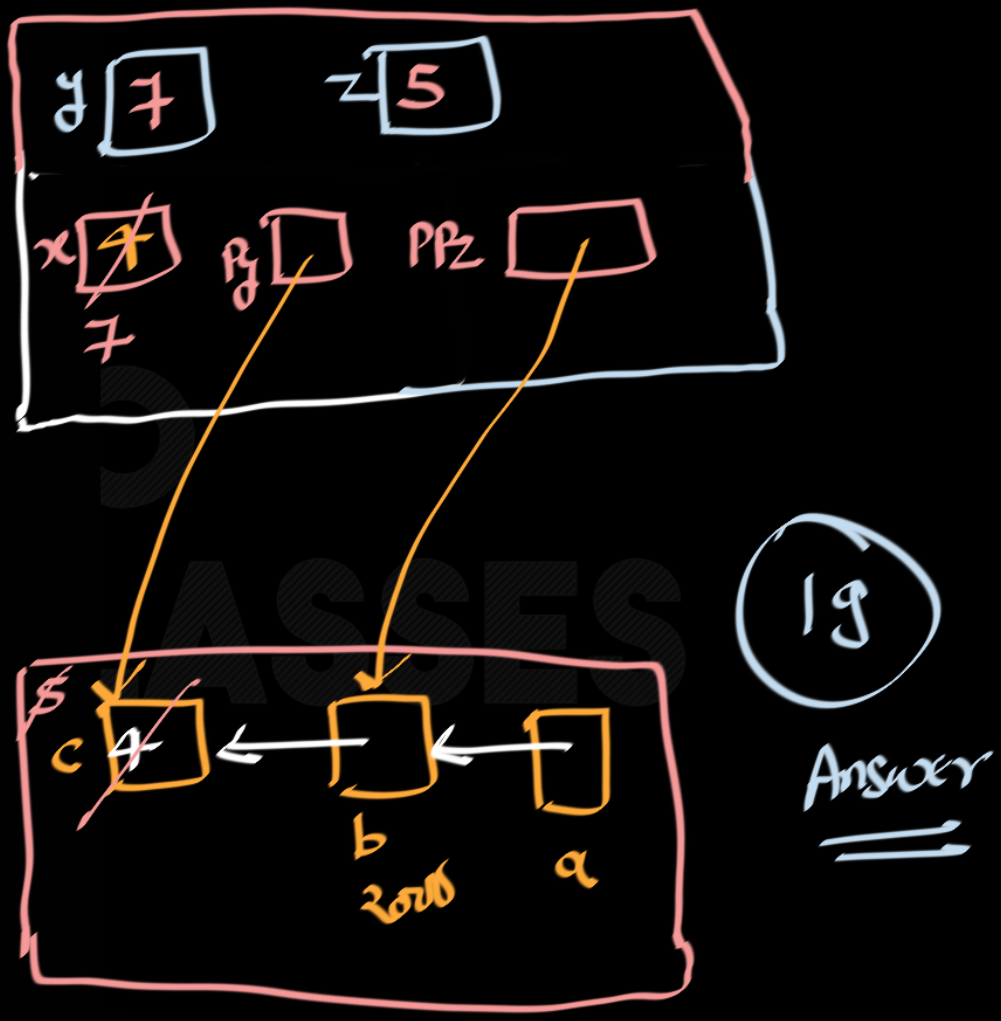
```
int f(int x, int *py, int **ppz)
{
    int y, z;
    **ppz += 1; z = **ppz;
    *py += 2; y = *py;
    x += 3;
    return x+y+z;
}

void main()
{
    int c, *b, **a;
    c = 4; b = &c; a = &b;
    printf("%d", f(c, b, a));

}
```

```c
int f(int x, int *py, int **ppz)
{
    int y, z;
    **ppz += 1; z = **ppz;
    *py += 2; y = *py;
    x += 3;
    return x+y+z;
}

void main()
{
    int c, *b, **a;
    c = 4; b = &c; a = &b;
    printf("%d", f(c, b, a));

}
```



**ppz += 1 ≡ **pp2 = **pp2 + 1

```c
int f(int x, int *py, int **ppz)
{
    int y, z;
    **ppz += 1; z = **ppz;
    *py += 2; y = *py;
    x += 3;
    return x+y+z;
}

void main()
{
    int c, *b, **a;
    c = 4; b = &c; a = &b;
    printf("%d", f(c, b, a));
}
```



**ppz += 1 ≡ **pp2 = **pp2 + 1

Question 4

D. What is the content of array `A` after executing the following code snippet?

```
long A[3] = {1, 2, 3}
long *p;
long **q;
p = A;
p++;
q = &p;
p++;
(*p) = (**q)*2;
```

1. 1 2 3
2. 1 2 4
3. 1 2 6
4. 1 4 3
5. 1 4 6
6. None of the above

Question 5

```
void inc_ptr(int **h)
{    *h = *h + 1;    }

int A[3] = {50, 60, 70};
int* q = A;
inc_ptr(&q);
printf("*q = %d\n", *q);
```

Question 6

```
int x[] = { 2, 4, 6, 8, 10 };
int *p = x;
int **pp = &p;
(*pp)++;
(*(*pp))++;
printf("%d\n", *p);
```

Result is:

A: 2

B: 3

C: 4

D: 5

https://inst.eecs.berkeley.edu/~cs61c/sp18/lec/03/lec03.pdf

Question 7

```c
#include <stdio.h>

void foo(int **p)
{
    int j = 11;
    *p = &j;
    printf("%d", **p);
}

int main()
{
    int i = 10;
    int *p = &i;
    foo(&p);
    printf("%d", *p);
    return 0;
}
```

Question 8

```
int i=10;
int *p;
int **q;
int ***r;
p=&i;
*p=15;
q=&p;
**q=20;
r=&q;
***r=(*p) + 1;
printf("%d",i);
```

Question 9

Consider the following code:

```
int a = 4;
int b = 6;
int c = 9;
int* p = &a;
int* q = p;
p = &b;
a++;
(*q)++;
b = *q * 2;
c = *q + *p;
```

What are the final values of a,b,c,p, and q?

Question 10

The correct prototype of a function `fun` that takes pointer to a `float`, a pointer to a pointer to a `char` and returns a pointer to a pointer to an `int` is (circle one answer):

```
int **fun(float**, char**);

int *fun(float*, char*);

int **fun(float*, char**);

int ***fun(*float, **char);
```

Question 11

```
void ubswap(int **a, int **b) {
    int* temp = *a;
    *a = *b;
    *b = temp;
}
int main() {
    int x = 1, y=9;
    int* u = &x; int* v = &y;
    int** a = &u; int** b = &v;
    ubswap(a, b);
    return 0;
}
```

which pairs of variables in `main` are swapped? Check all that apply.

☐ `x and y` ☐ `u and v` ☐ `a and b` ☐ None of the above

https://cse.buffalo.edu/~hungngo/classes/2014/Fall/250/assignments/sample-midterm1-sol.pdf

Answer

```
void ubswap(int **a, int **b) {
    int* temp = *a;
    *a = *b;
    *b = temp;
}
int main() {
    int x = 1, y=9;
    int* u = &x; int* v = &y;
    int** a = &u; int** b = &v;
    ubswap(a, b);
    return 0;
}
```

which pairs of variables in `main` are swapped? Check all that apply.

☐ `x and y` ⊗ `u and v` ☐ `a and b` ☐ None of the above

Question 12

```c
int main(void) {
  int arr[3];

  for (int i = 0; i < 3; i++) {
    arr[2 - i] = 2*i;
    printf("arr[%u]=%u\n", 2 - i, 2 * i);
  }

  int *curr = &arr[0];
  int *next = NULL;

  curr = aFunction( &curr, &next);

  printf("*curr=%d\n", *curr);
  printf("*next=%d\n", *next);

  for (size_t i = 0; i < 3; i++) {
    printf("arr[%u]=%d\n", i, arr[i]);
  }
}
```

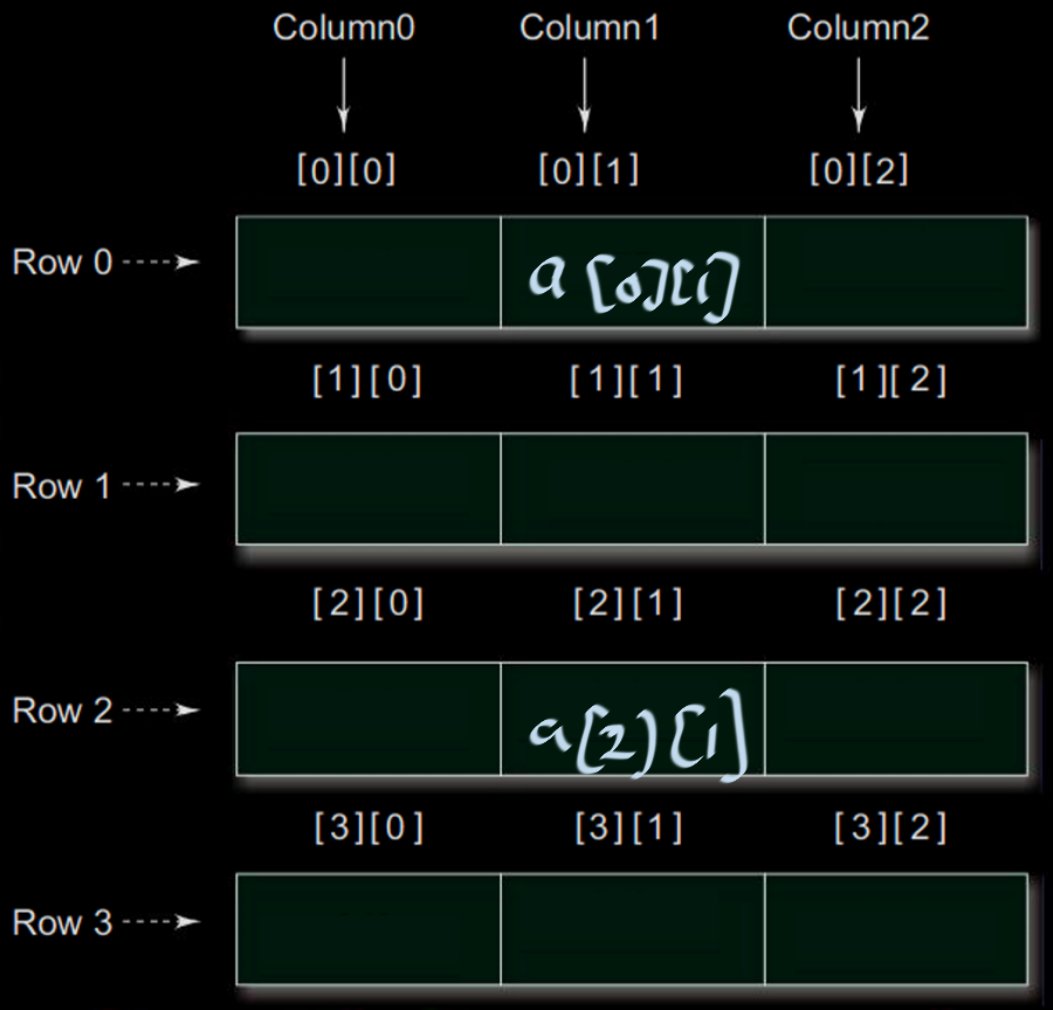
```c
int *aFunction(int **p, int **q)
{
  printf("**p=%d\n", **p);


  (*p)++;
  *q = *p + 1;

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

  **q = **p - (*p)[-1];

  int **temp = p;
  p = q;
  q = temp;


  return *q - 1;
}
```

2 D arrays

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

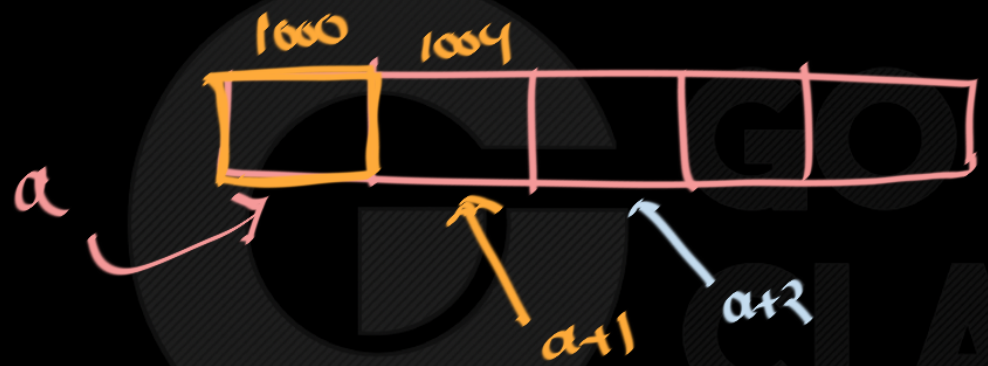
No. of rows

No. of columns



GO CLASSES

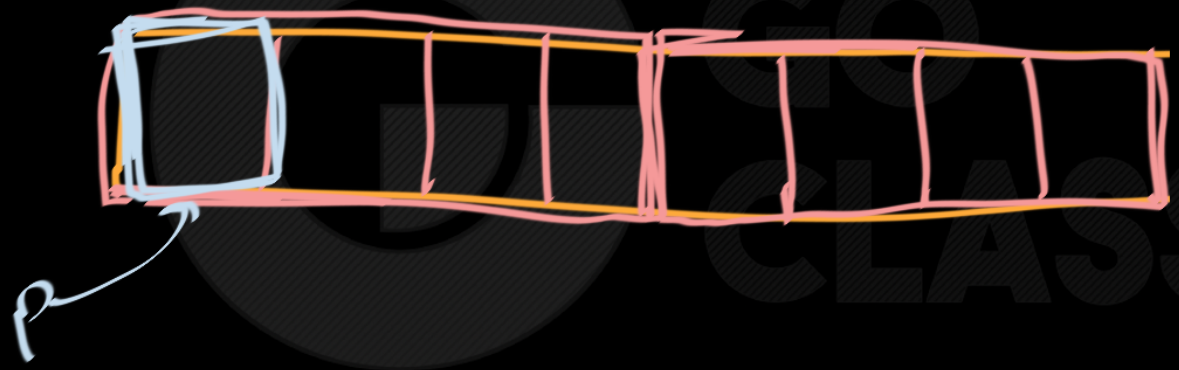
1 D array



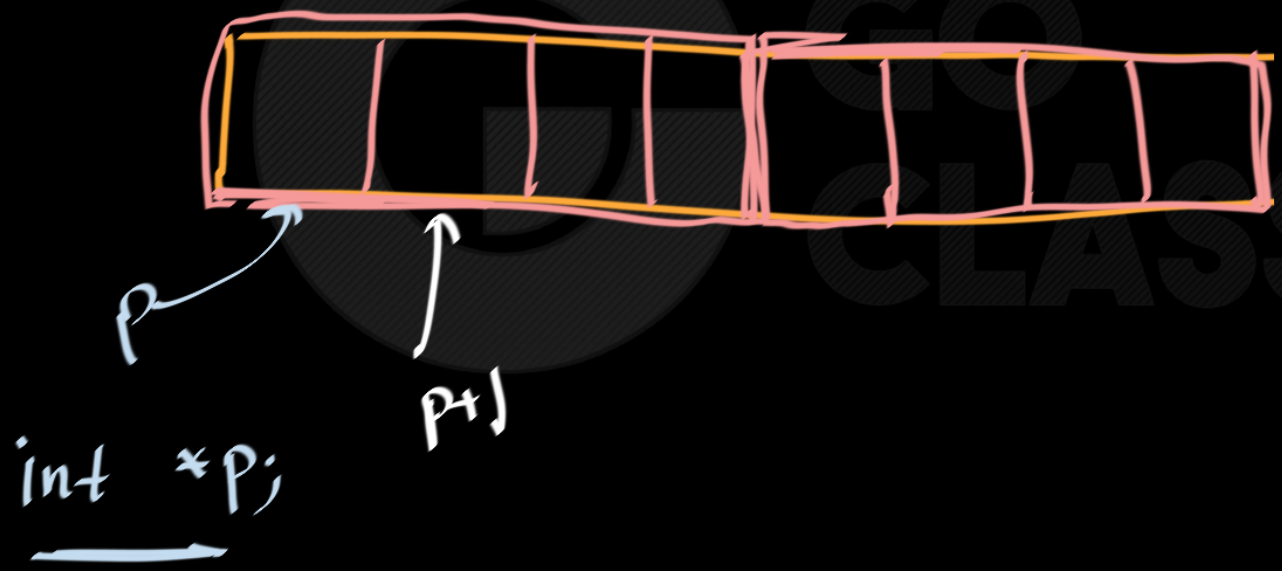
int a[5];

a ≡ 1000

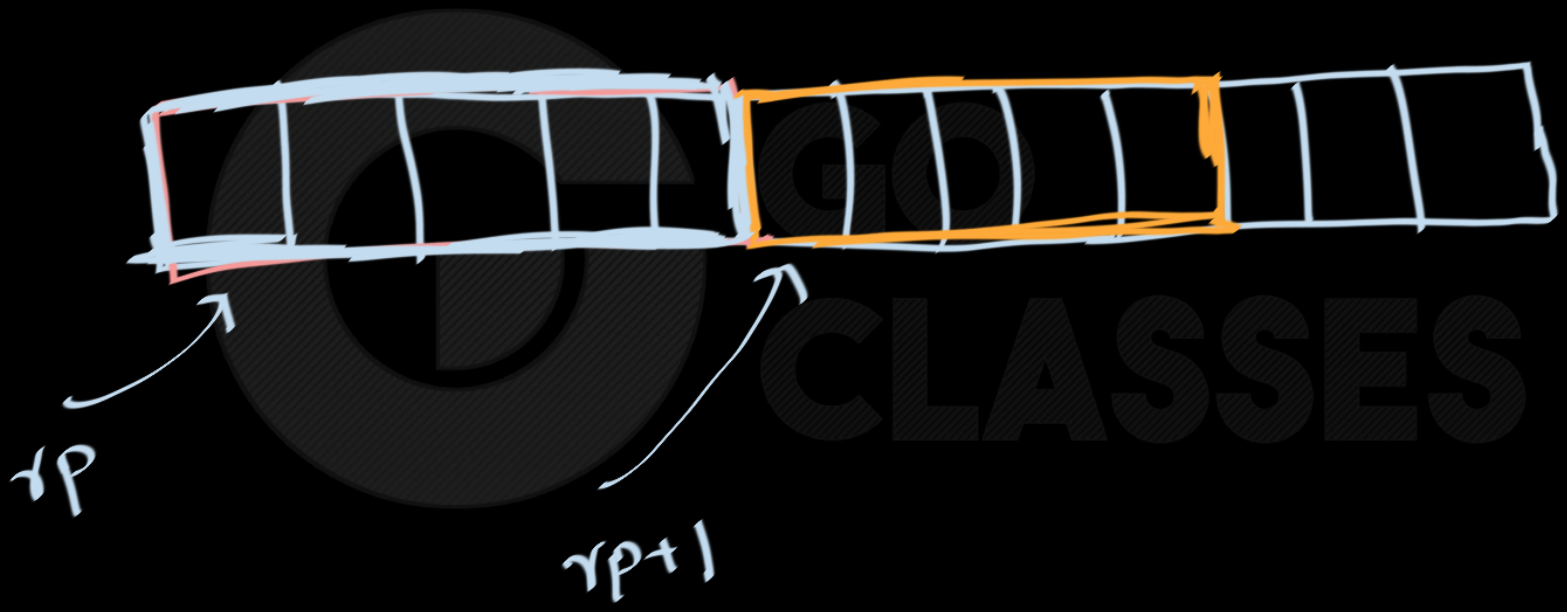
a+1 ≡ 1004



int *p;

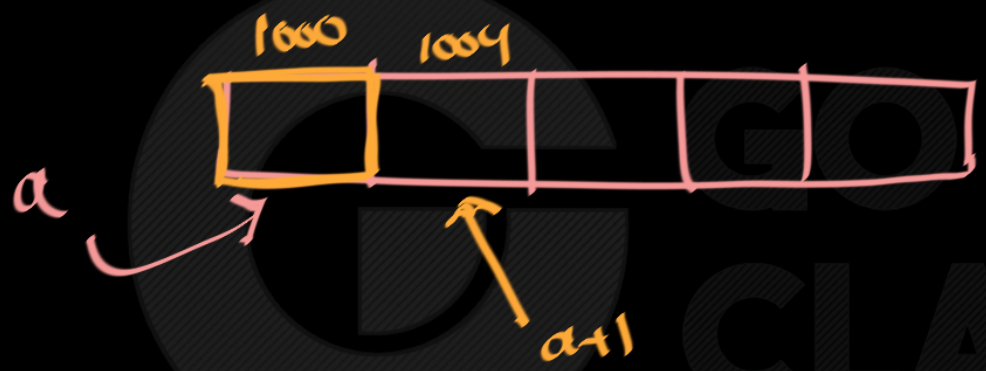

p
p+1
int *p;


```
int *rp;      Wrong

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


rp
rp+1

GO CLASSES

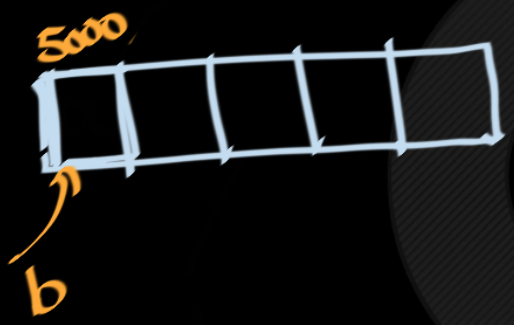
1D array



a ≡ 1000

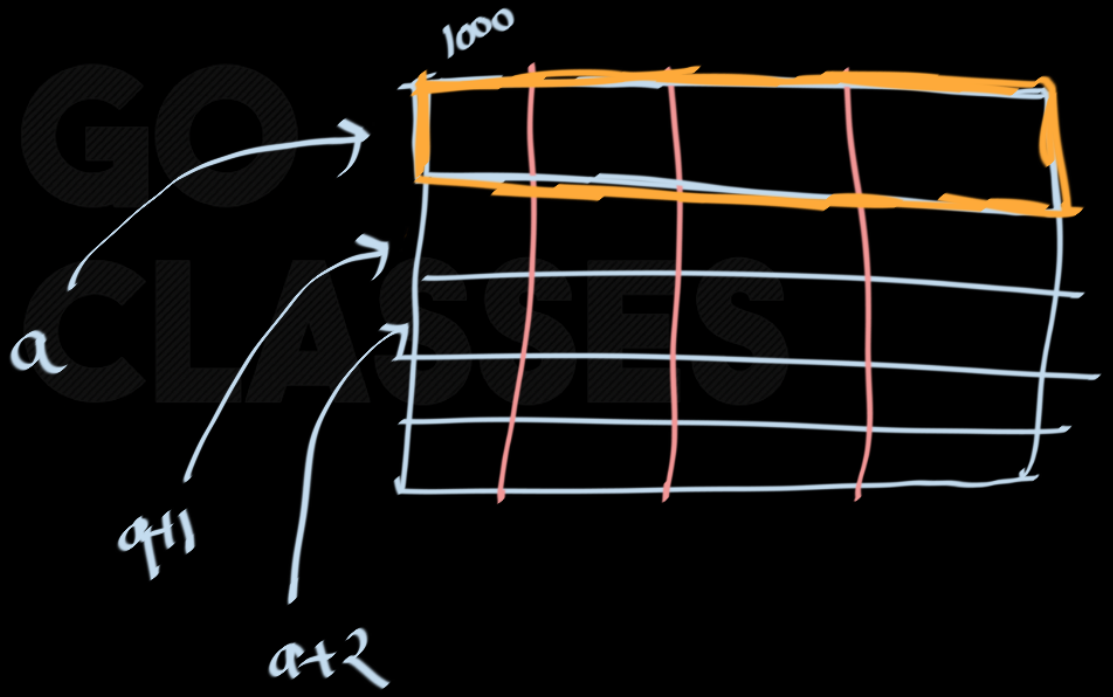
GO CLASSES

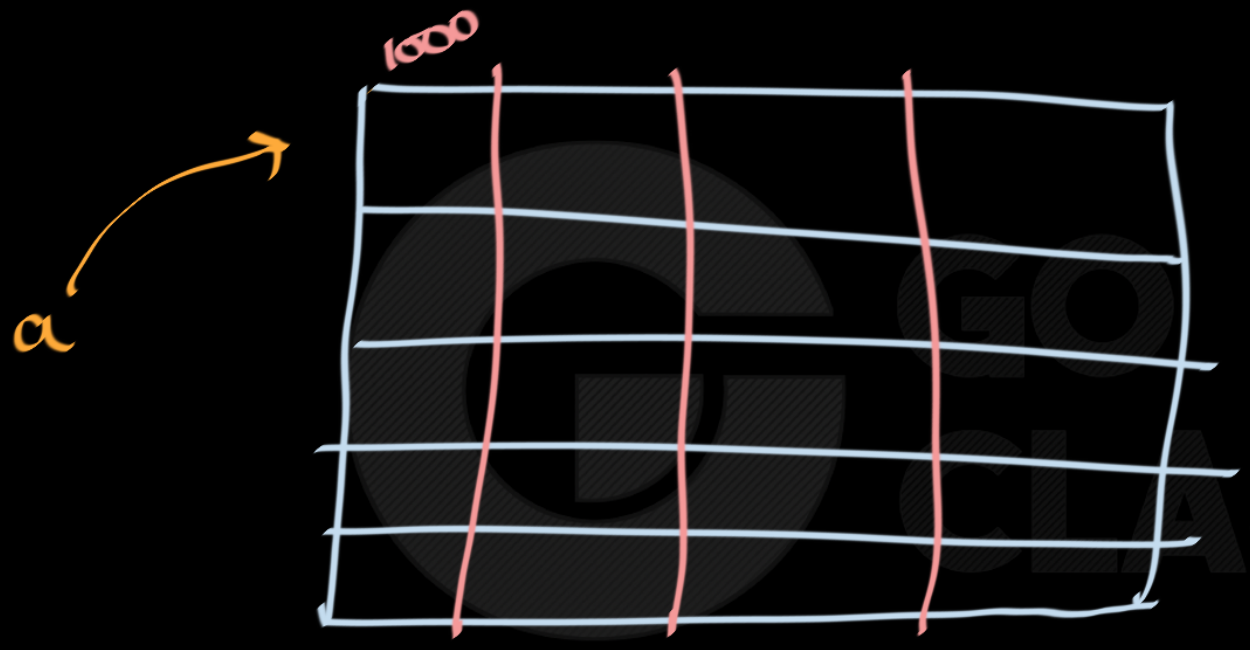
int b[5];



b ≡ 5000

int a[5][4];



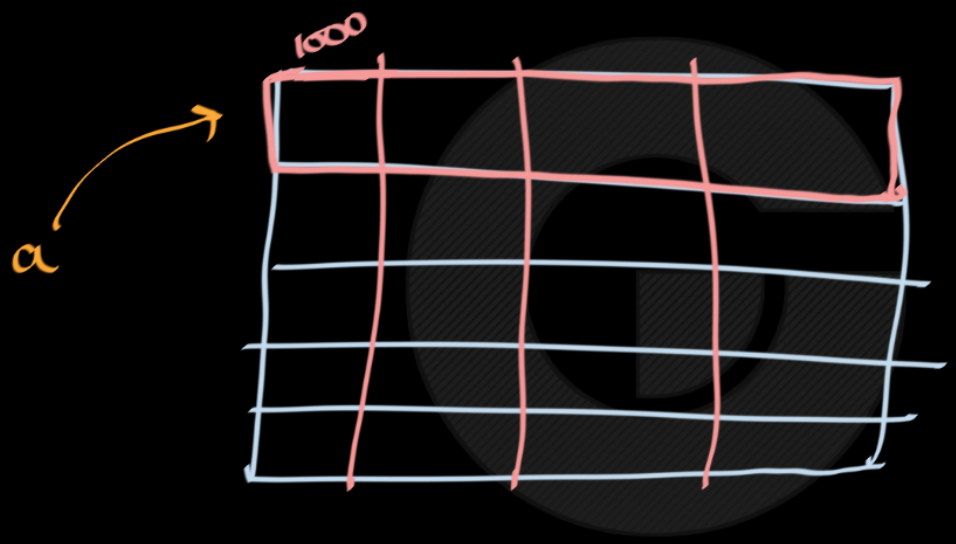


int a[5][4];

a ≡ 1000

a : Points to the very first row

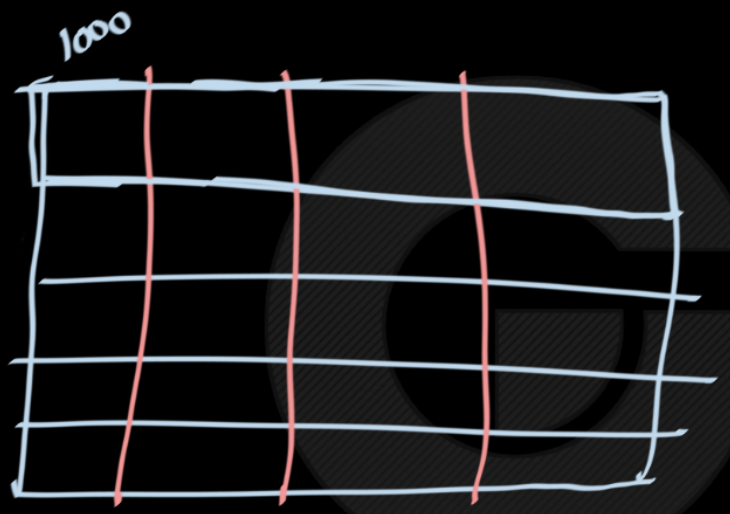
int a[5][4];



a : Points to the very first row

; is an address of first row

≡ 1000



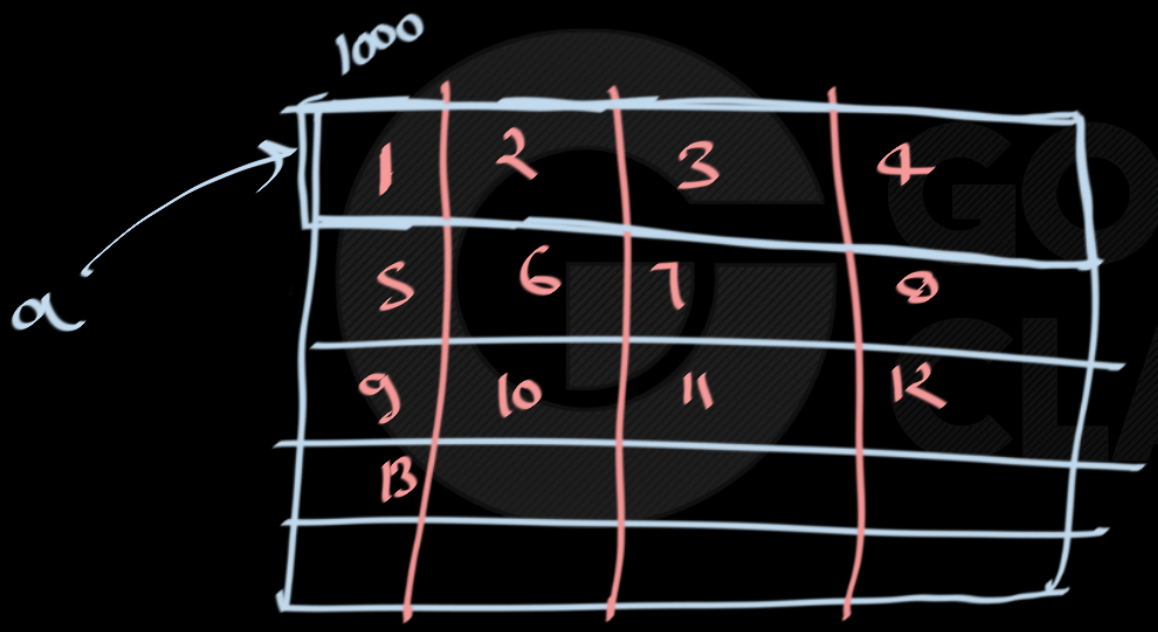
1000 is ⇒

(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.

GO CLASSES

int b [5]

b → address / pointer

b[3] → integer

*(b+1) ⇒ integer

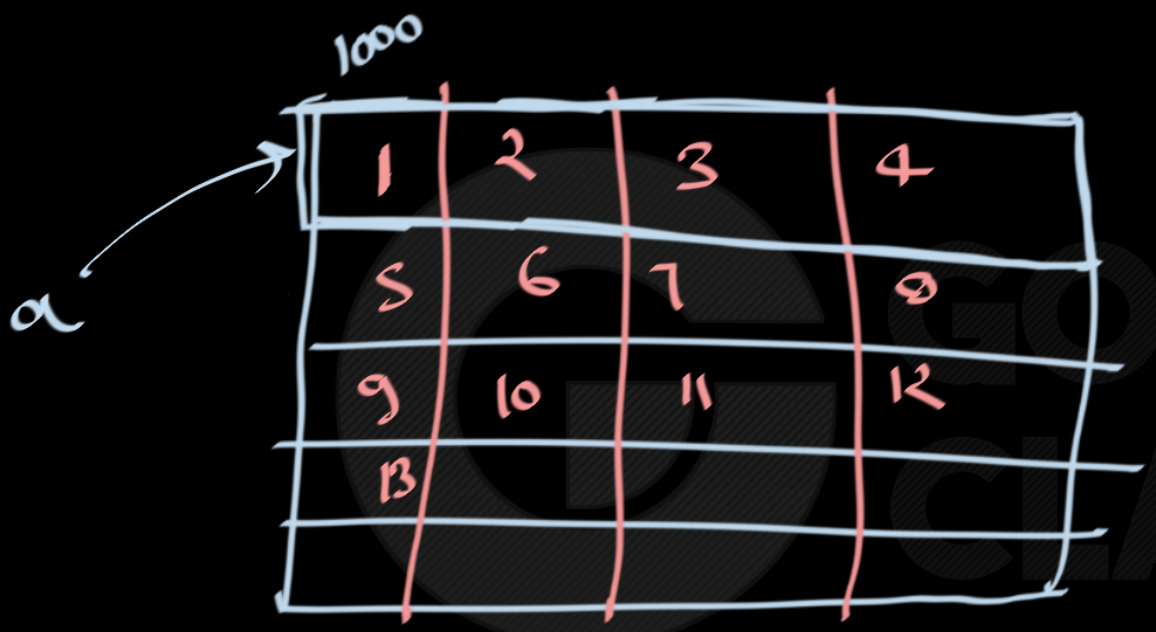
int a [3] [4]

[ ] [ ] } integer

[ ] *

* *

a[0] [1] ↙ *a[1] ↙ *(*(a+1)+2)

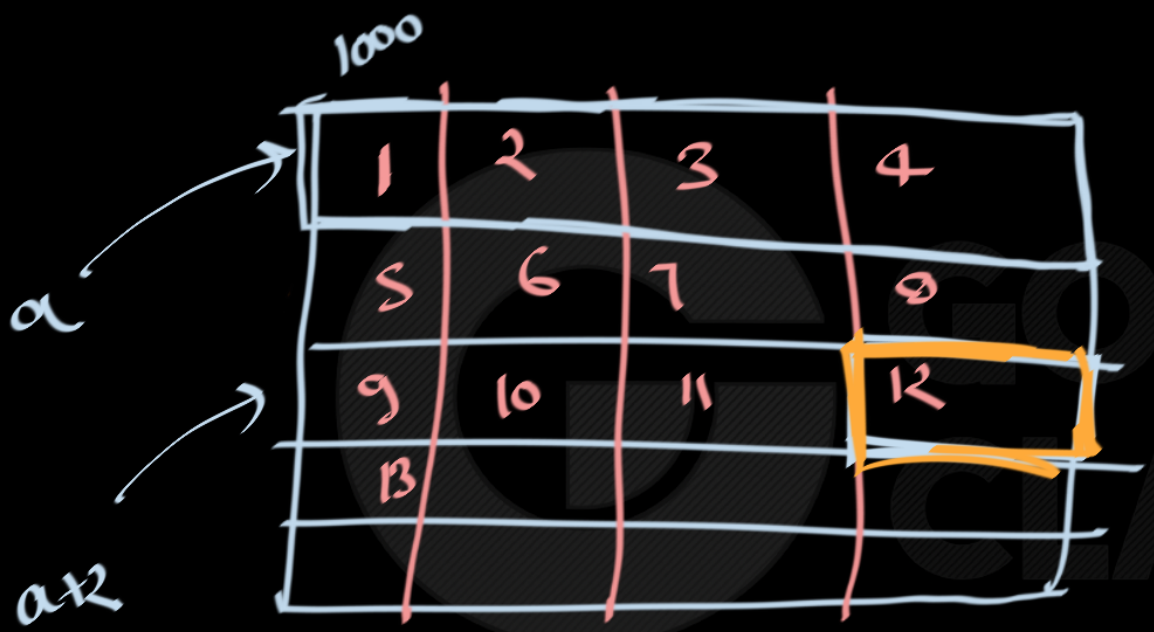


$a \equiv 1000$ ← add. of 1st row

$*a \equiv 1000$ ← 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

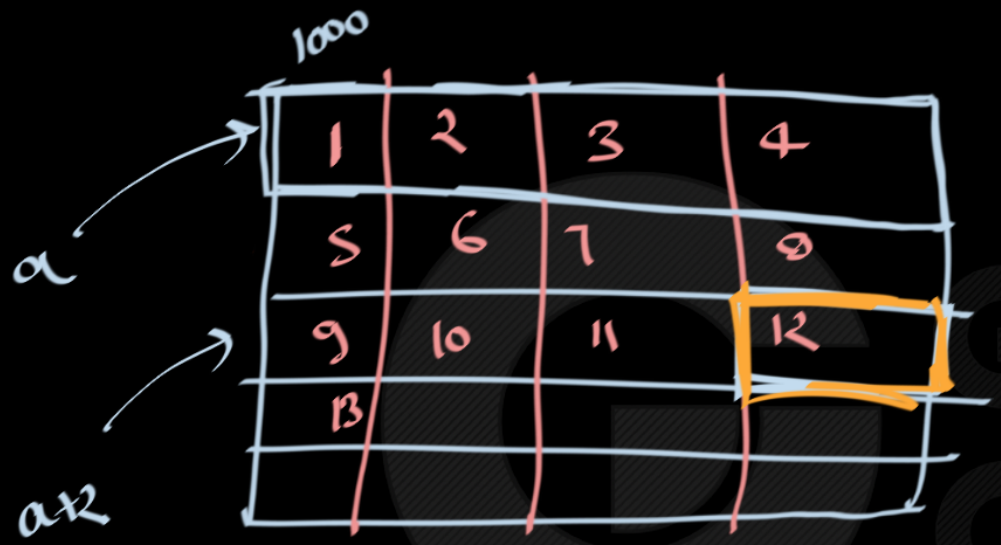
```
int b[5];
```

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

in 2D array

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

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



$a[2][3]$

or

$\equiv R$

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

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

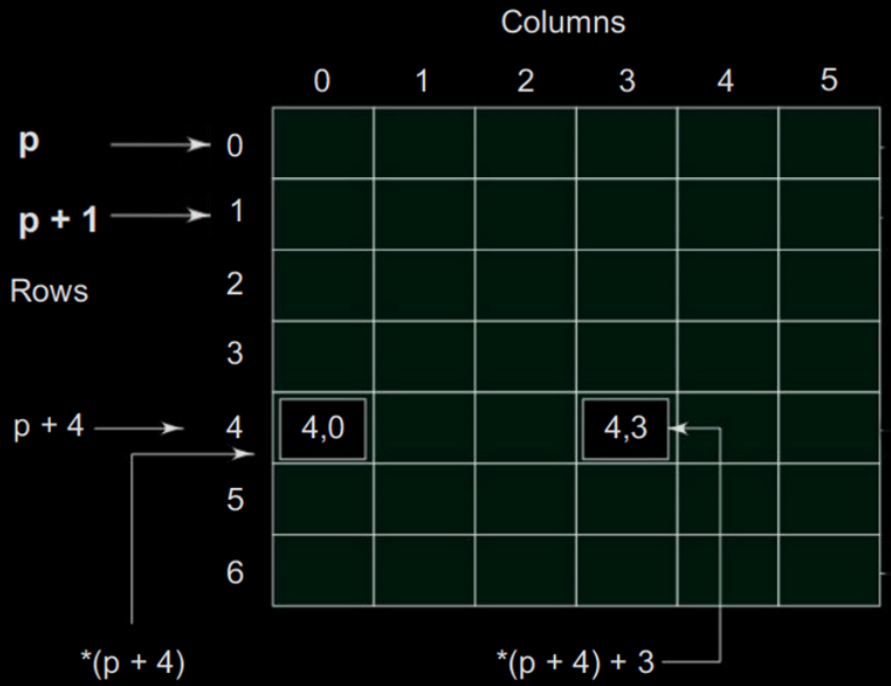
$a[1][7] \equiv R$

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

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

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

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



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}};
```

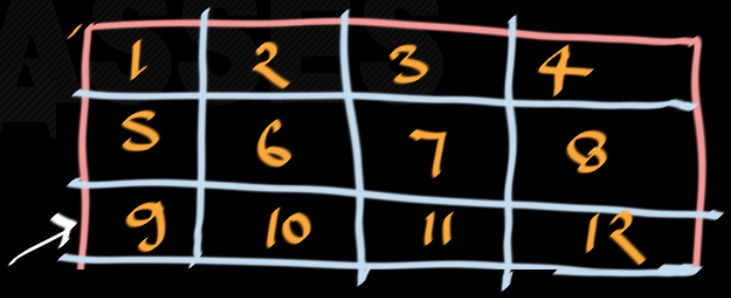
Which of the following print "7" ?

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

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

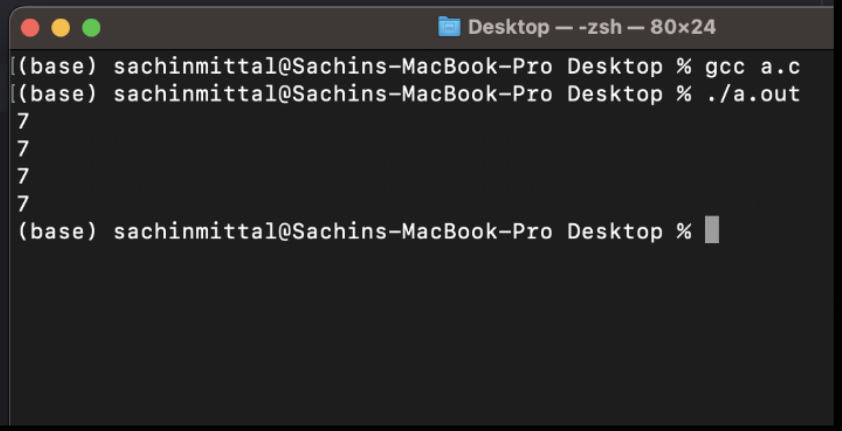
C. `printf("%d", *(*x+6));`

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];

              }
        }
}
```