Pointers in C

Allows us to indirectly access variables

A pointer:

- Contains an address
- Allows the memory at that address to be manipulated
- Associates a type with the manipulated memory

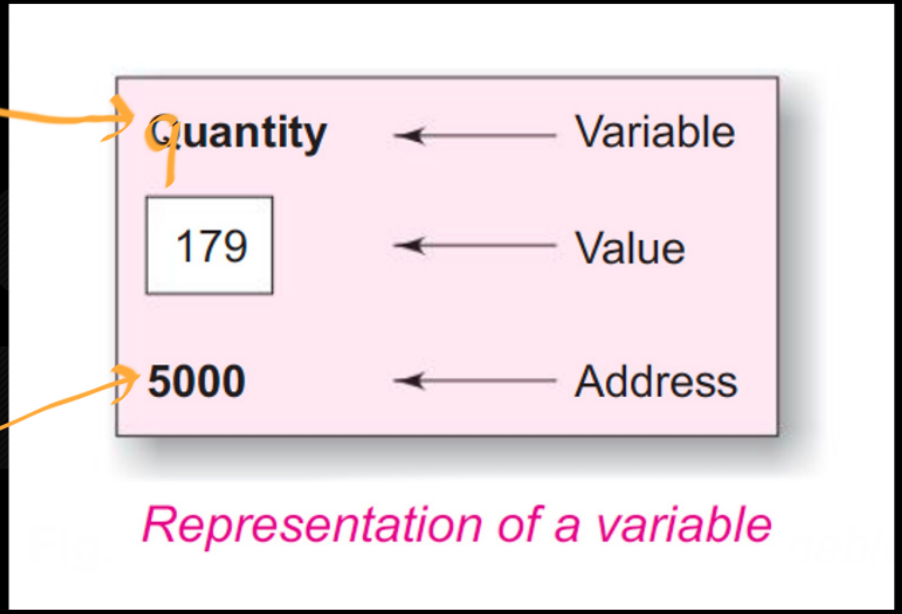
```
int quantity = 179 ;
```

House Name

This is something similar to having a house number as well as a house name.

Pointers is, nothing but a variable that contains an address.

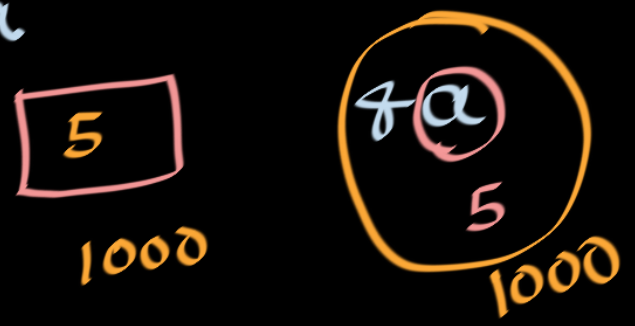
House Number



Representation of a variable

```
Pf ("%d", quantity)
```

GO CLASSES



int a = 5;

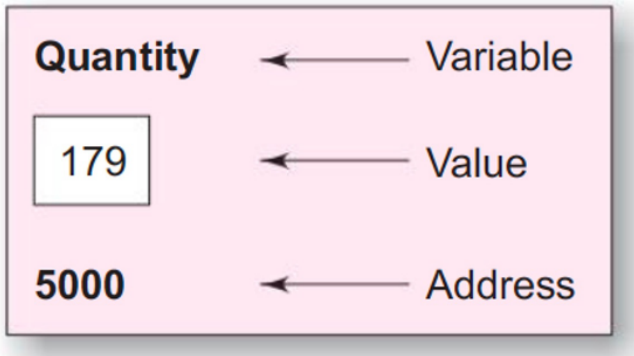
this is a variable which holds integer value

Pointer

int *p = &a

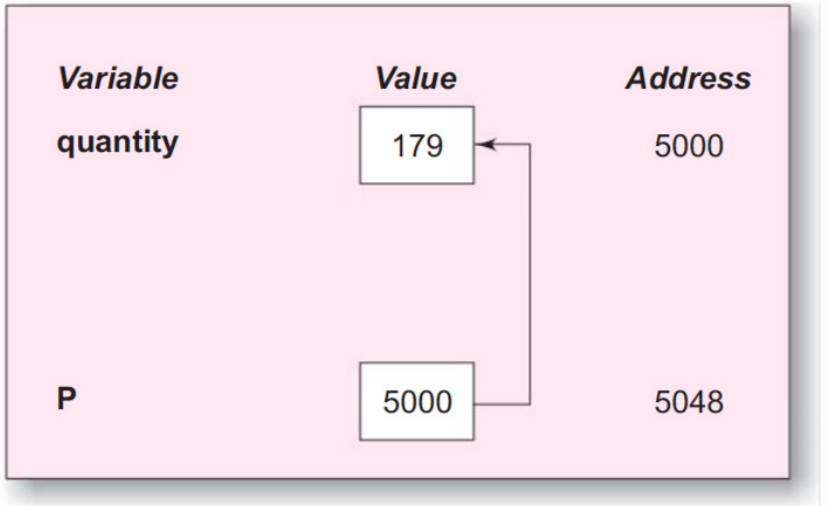
it is a variable, which holds address of int

```
int quantity = 179;
```

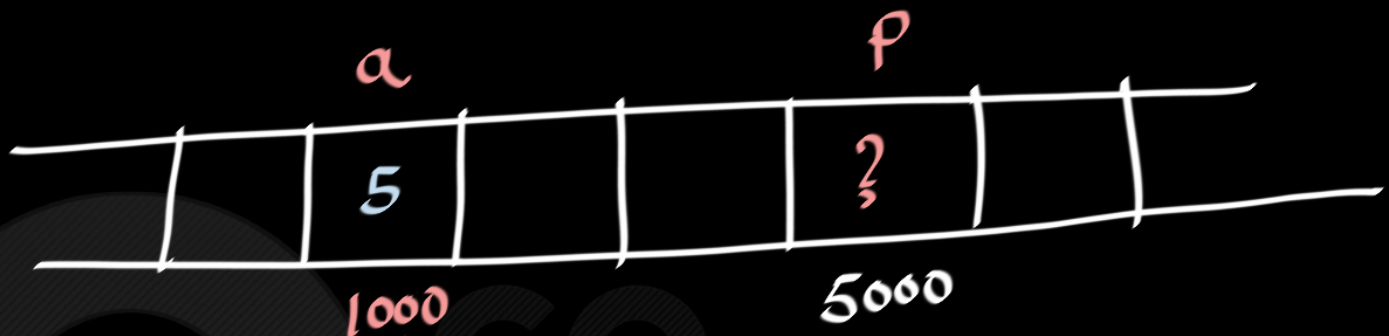


Representation of a variable

```
p = &quantity;
```



Pointer variable



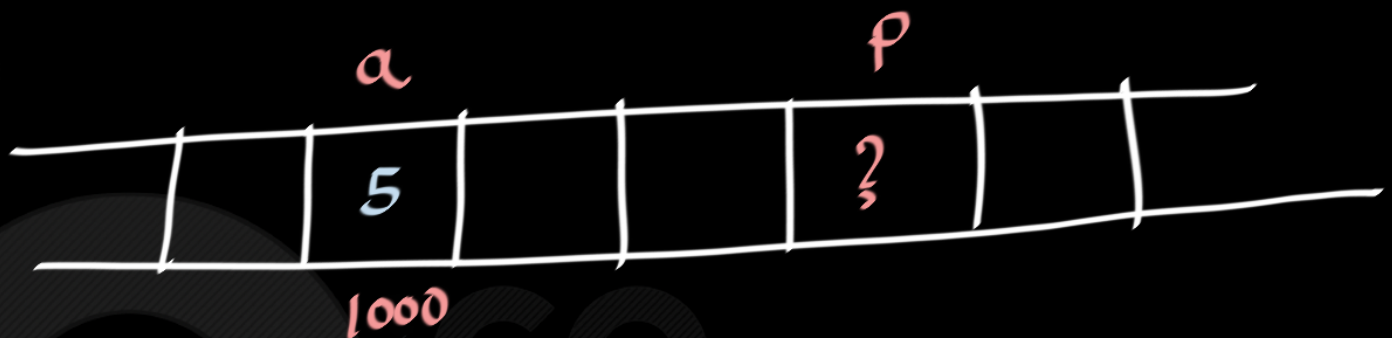

```
int  a = 5;
int *p;
char c;
```

is "a" a variable? Yes

is "p" a variable? yes

is "c" a variable? Yes

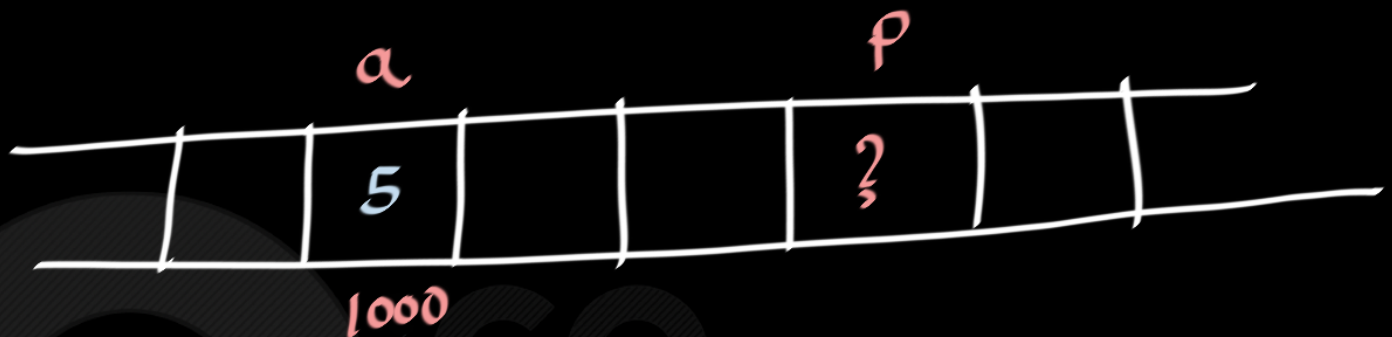
it is a variable which holds an integer



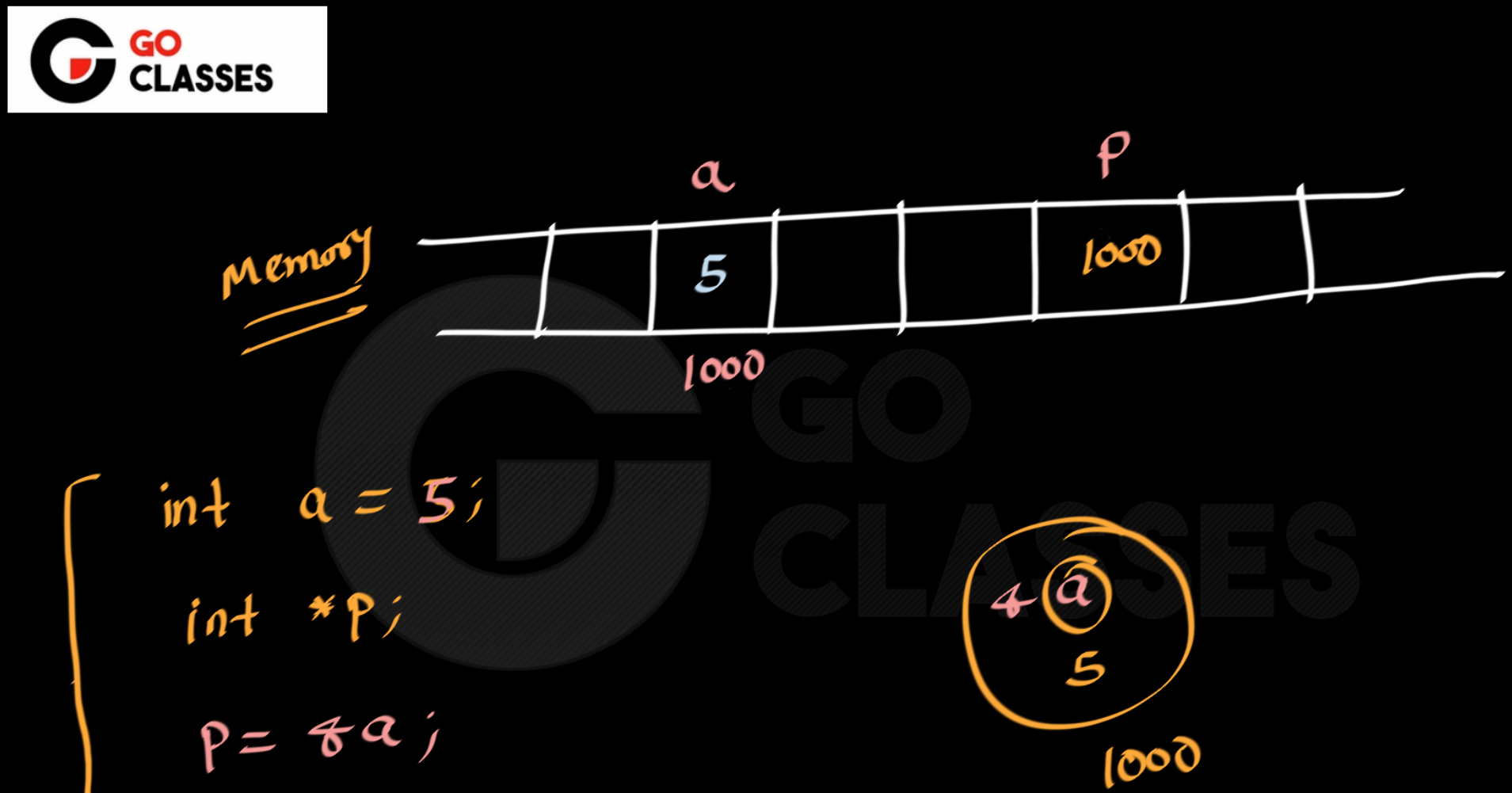

```
int  a = 5;

int *p;
```

it is a variable which holds an address of integer.



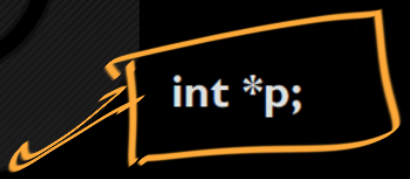

```
int  a = 5;
int *p;
```

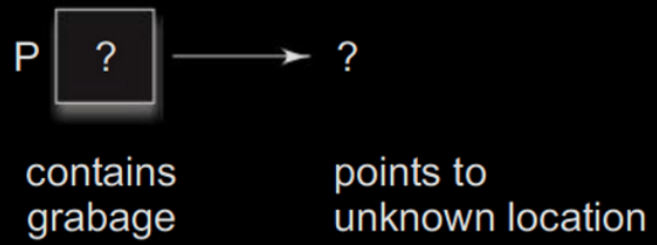

a
p
Memory
5
1000
1000
int a = 5;
int *p;
p = &a;
4
a
5
1000

```
int *p; /* integer pointer */

float *x;  / * float pointer */
```

Creating Pointer Variable





Step by Step, show memory changes.

```
int *p, x;

x = 3;

p = &x;
```

Step by Step, show memory changes.

```
int *p, x;

x = 3;

p = &x;
```

Step by Step, show memory changes.

```
int *p, x;

x = 3;

p = &x;
```

Step by Step, show memory changes.

```
int *p, x;
x = 3;
p = &x;
```

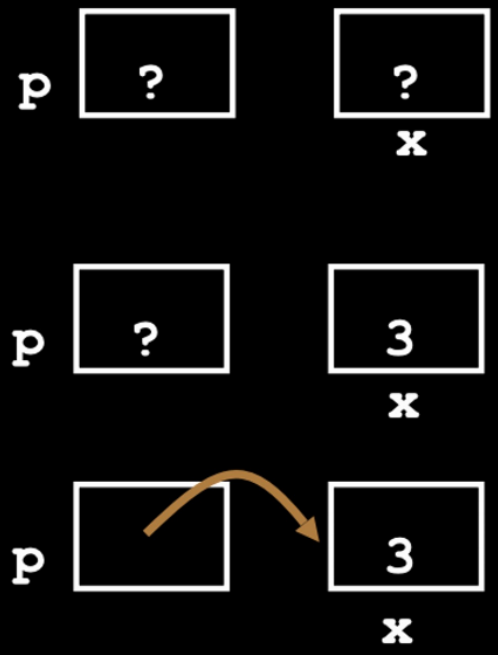
Step by Step, show memory changes.

```
int *p, x;
x = 3;
p = &x;
```

```
int *p, x;

x = 3;

p = &x;
```



Pointer Declaration Style

```
int*    p;    /* style 1 */
int     *p;   /* style 2 */
int     * p;  /* style 3 */
```

✓ Preferred

Illegal uses

- &125 (pointing at constants).

- &(x+y) (pointing at expressions).

- ```
  register int x;
  int *p = &x;
  ```
  ```

Illegal uses

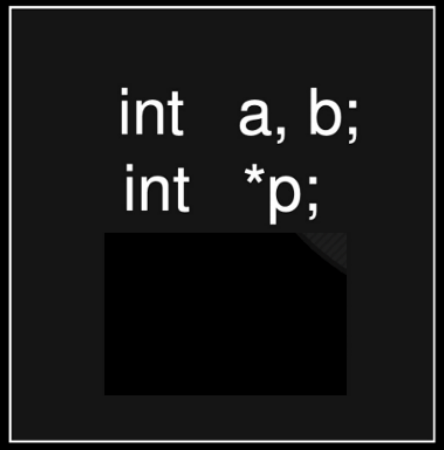
you cannot take an address of register variable,

even if compiler decides to keep variable in memory rather than in register.

```
register int x;
int *p = &x;
```

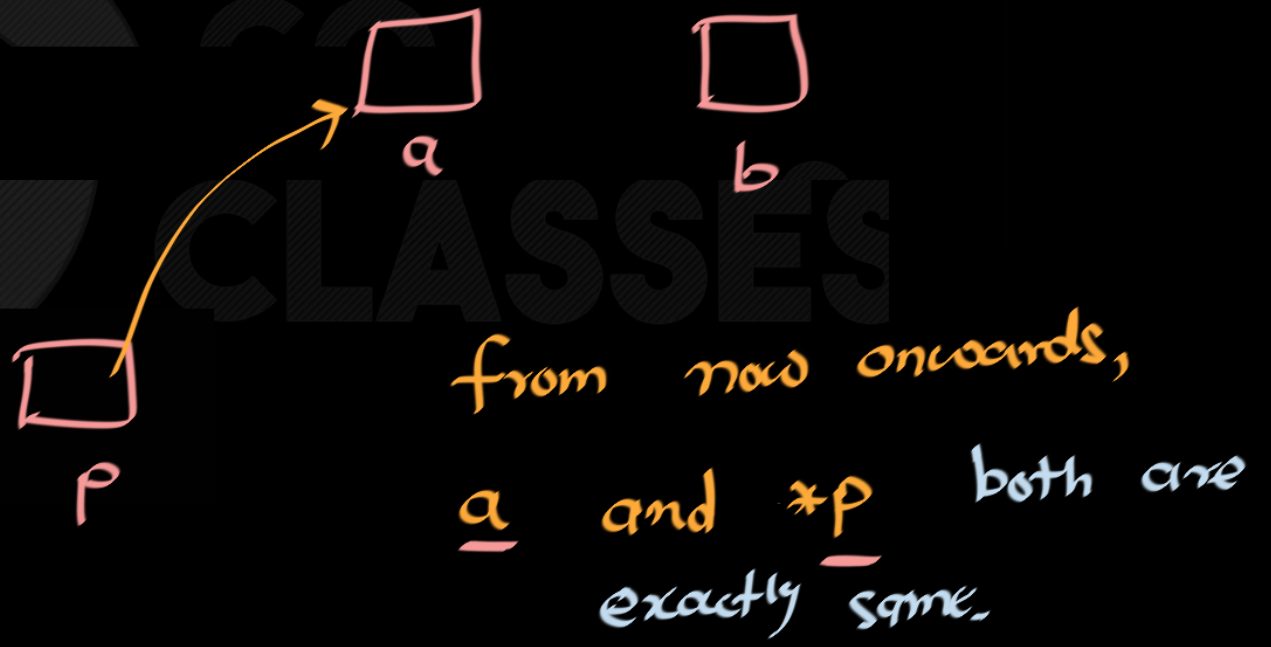
Accessing a Variable Through its Pointer

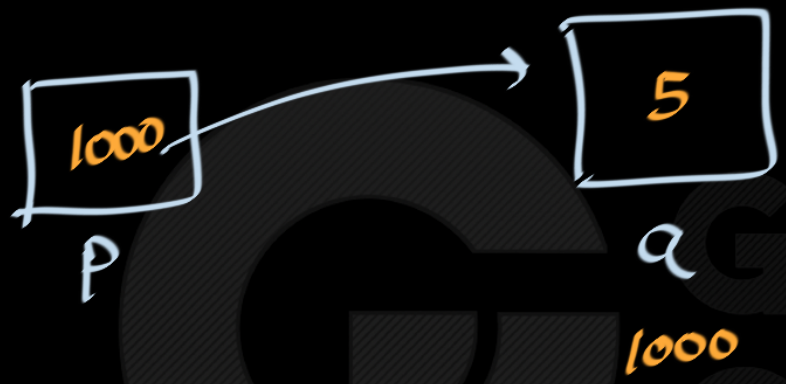
```
int   a, b;
int   *p;
```

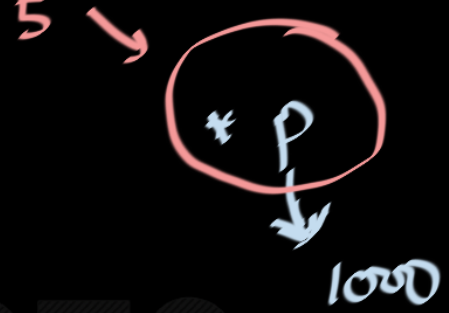


Accessing a Variable Through its Pointer

```
int   a, b;
int   *p;
p = &a;
```







p ⇒ 1000

a ⇒ 5

*p ⇒ 5

*a ⇒ 1000

Accessing a Variable Through its Pointer

```
int   a, b;
```

```
int   *p;
p = &a;
b = *p;
```

Equivalent to

```
b = a;
```

```
#include  <stdio.h>
int main()
{
    int    a, b;
    int    c = 5;
    int    *p;

    a   =   4   *   (c   +   5) ;

    p   =   &c;

}
```

from now onwards *p and c are same.

```
#include  <stdio.h>
int main()
{
    int    a, b;
    int    c = 5;
    int    *p;

    a  =  4  *  (c  +  5) ;

    p  =  &c;
    b  =  4  *  (*p  +  5) ;
    printf  ("a=%d  b=%d \n",  a, b);
    return 0;
}
```

Equivalent

p = &c

*p

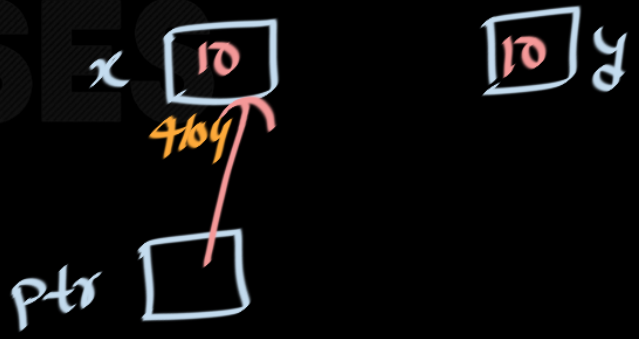
a=40 b=40

```
main()
{
    int x, y;
    int *ptr;
    x = 10;
    ptr = &x;
    y = *ptr;
    printf("Value of x is %d\n\n",x);
    printf("%d is stored at addr %u\n", x, &x);

}
```

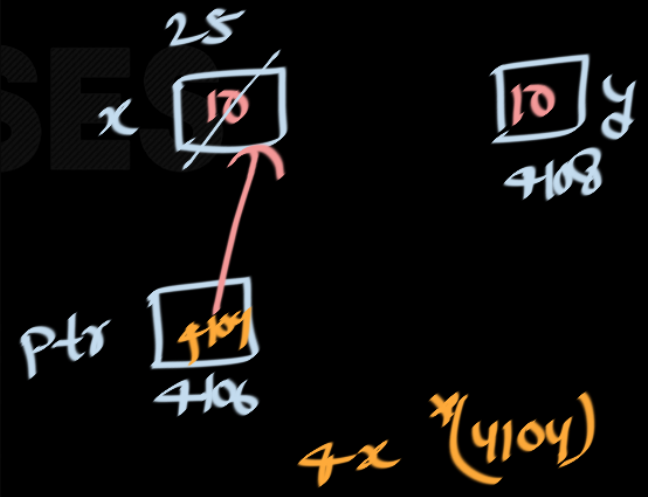
Output

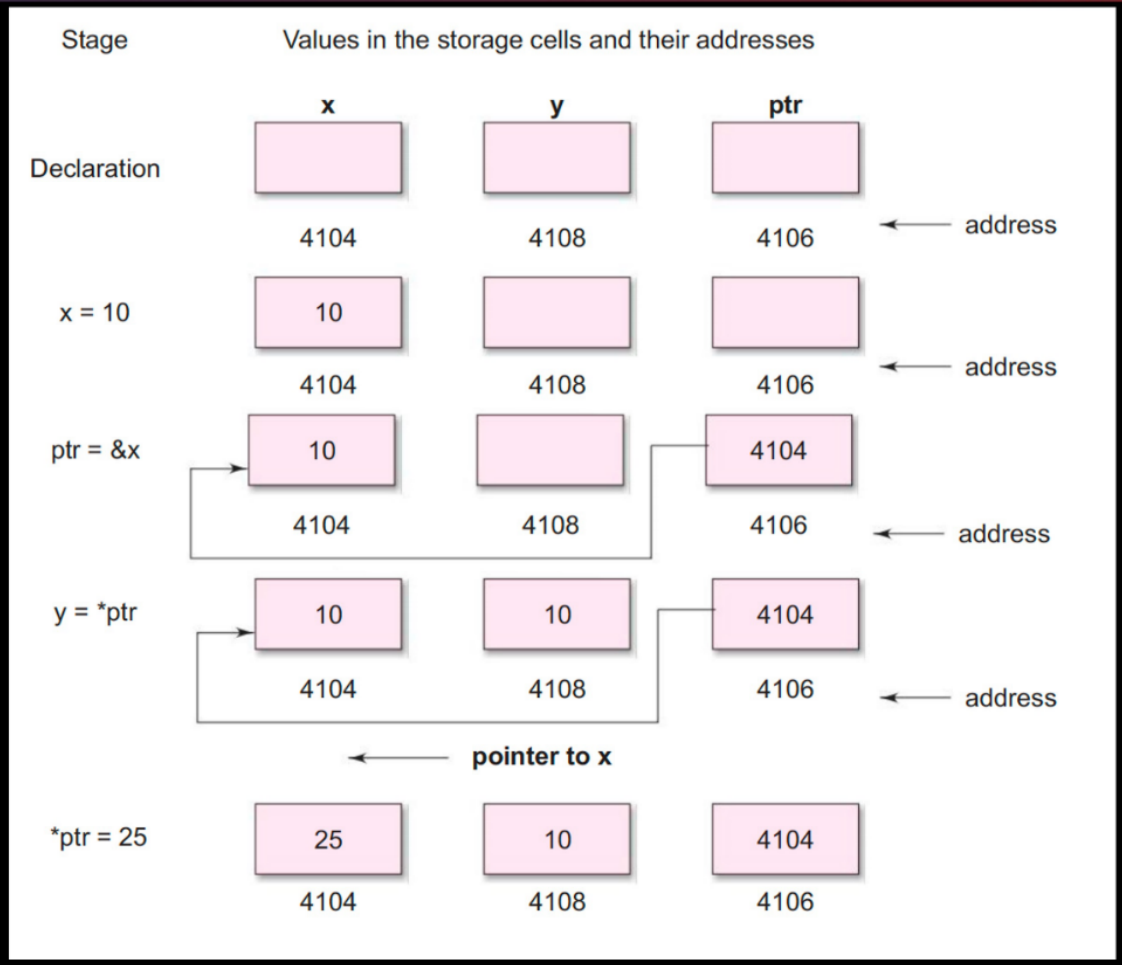
```
Value of x is 10
10 is stored at addr 4104
```



```c
main()
{
    int x, y;
    int *ptr;
    x = 10;
    ptr = &x;
    y = *ptr;
    printf("Value of x is %d\n\n",x);
    printf("%d is stored at addr %u\n", x, &x);
    printf("%d is stored at addr %u\n", *&x, &x);
    printf("%d is stored at addr %u\n", *ptr, ptr);
    printf("%d is stored at addr %u\n", ptr, &ptr);
    printf("%d is stored at addr %u\n", y, &y);
    *ptr = 25;
    printf("\nNow x = %d\n",x);
}
```

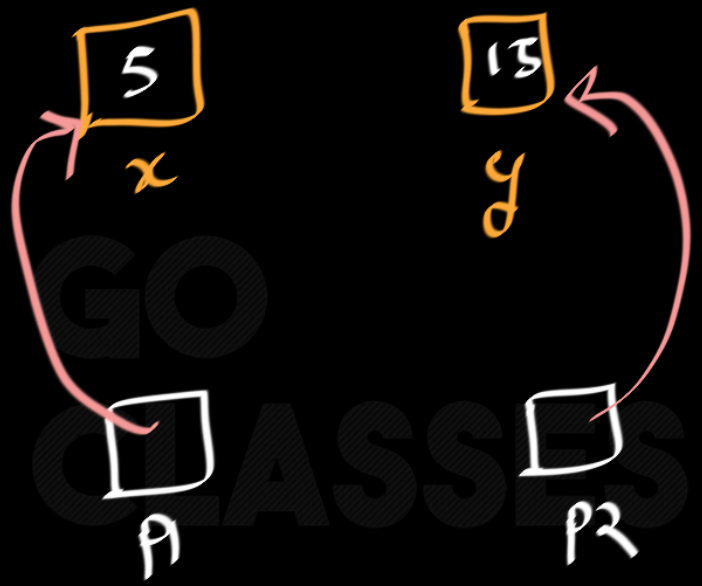
```
Output
Value of x is 10
10 is stored at addr 4104
10 is stored at addr 4104
10 is stored at addr 4104
4104 is stored at addr 4106
10 is stored at addr 4108
Now x = 25
```




Stage
Values in the storage cells and their addresses
x
y
ptr
Declaration
4104
4108
4106
address
x = 10
10
4104
4108
4106
address
ptr = &x
10
4104
4104
4108
4106
address
y = *ptr
10
10
4104
4104
4108
4106
address
pointer to x
*ptr = 25
25
10
4104
4104
4108
4106

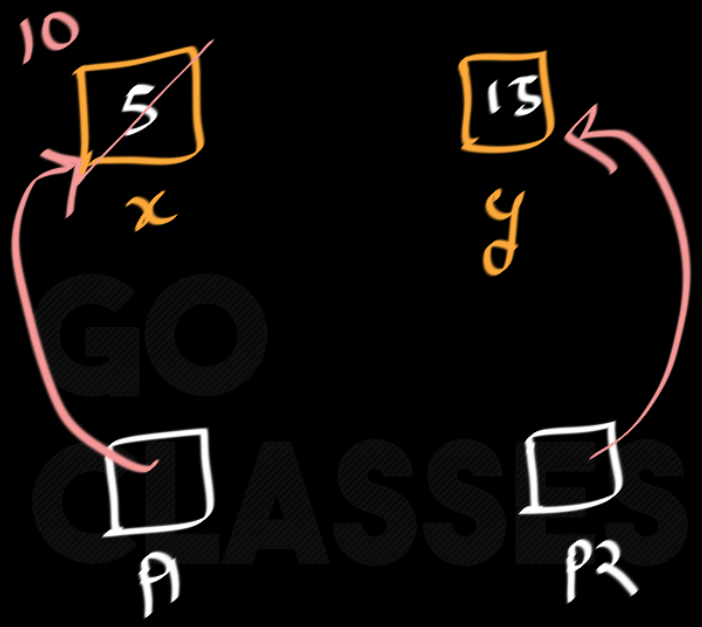
What is the output of the following code snippet?

```
1. int x = 5, y = 15;
2. int * p1, * p2;
3. p1 = &x;
4. p2 = &y;
```



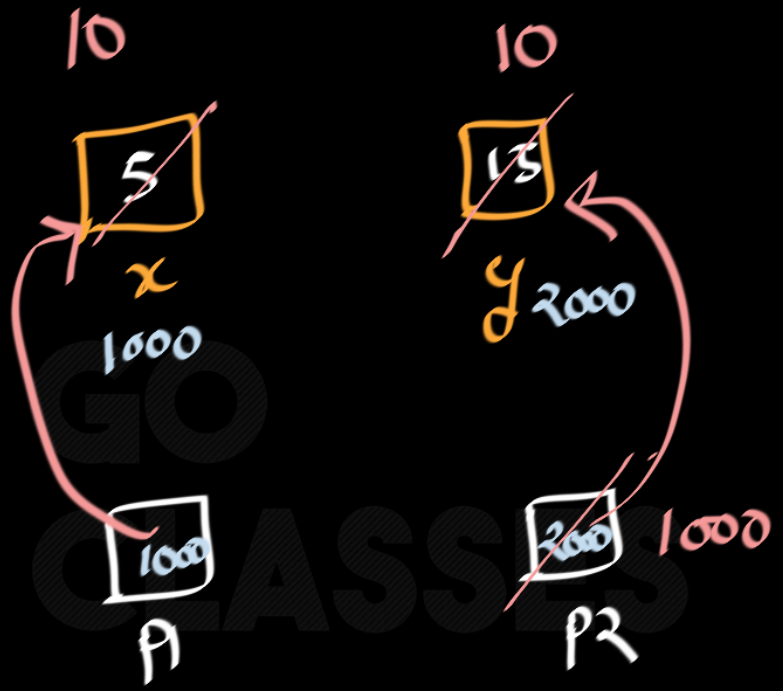
What is the output of the following code snippet?

1. int x = 5, y = 15;
2. int * p1, * p2;
3. p1 = &x;
4. p2 = &y;
5. *p1 = 10;



What is the output of the following code snippet?

1. int x = 5, y = 15;
2. int * p1, * p2;
3. p1 = &x;
4. p2 = &y;
5. *p1 = 10;
6. *p2 = *p1;

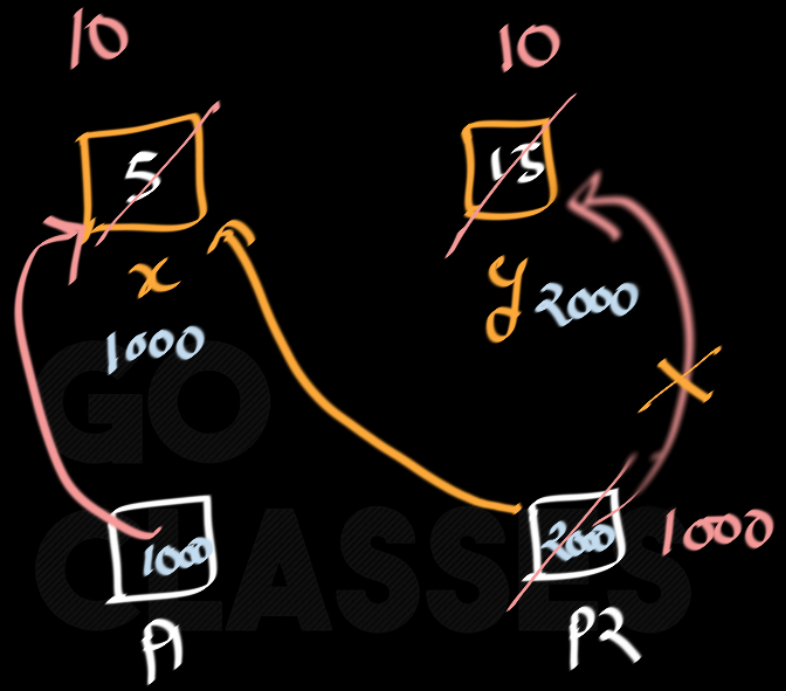


y = x

p2 = p1

What is the output of the following code snippet?

```
1. int x = 5, y = 15;
2. int * p1, * p2;
3. p1 = &x;
4. p2 = &y;
5. *p1 = 10;
6. *p2 = *p1;
```

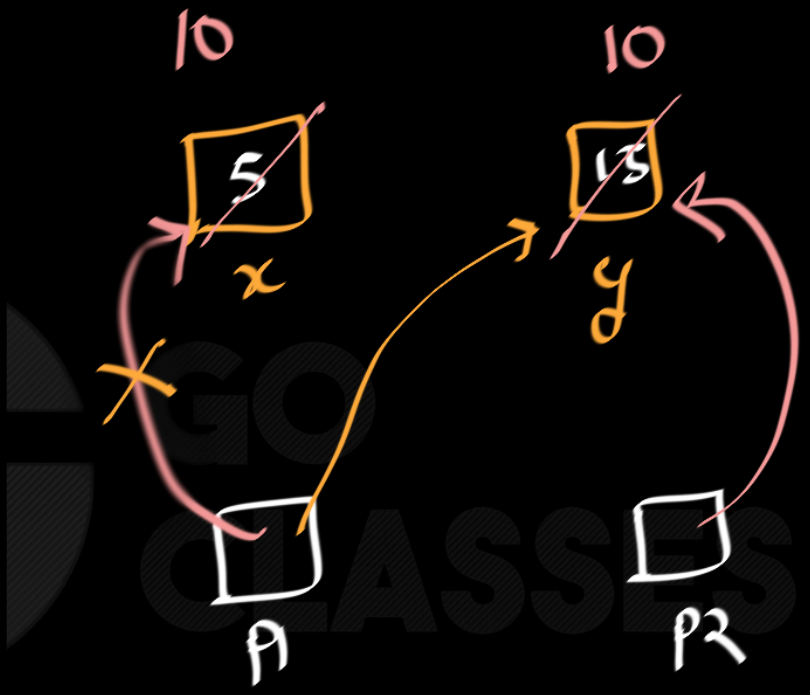


$y = x$

$p2 = p1$ ⇒ this line means p2 will start pointing to whatever p1 is pointing to.

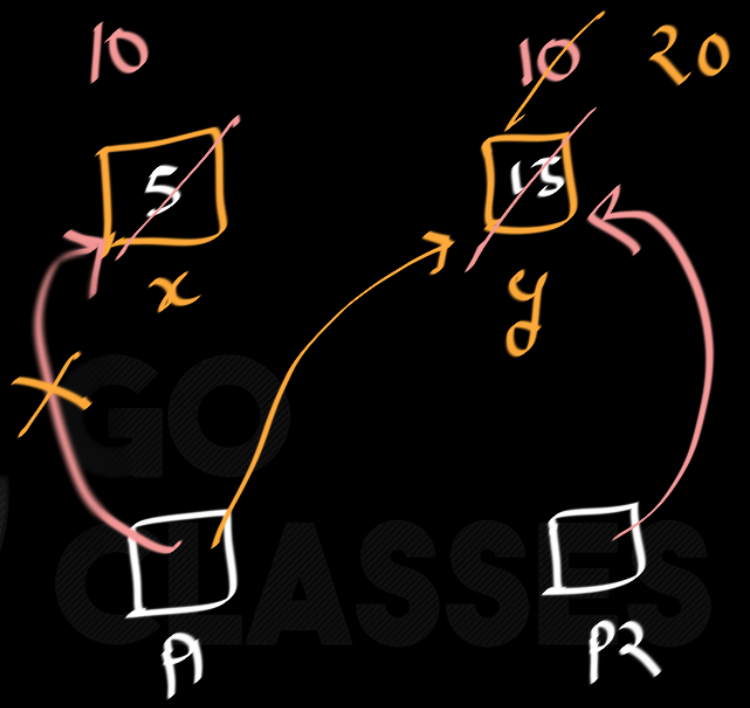
What is the output of the following code snippet?

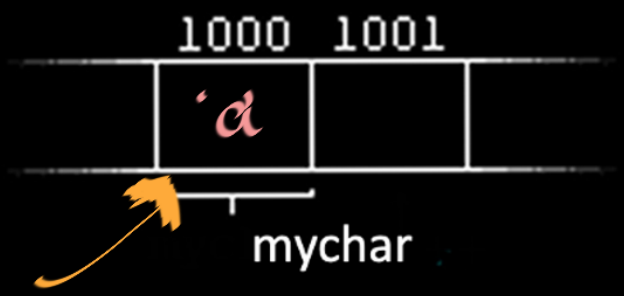
```
1. int x = 5, y = 15;
2. int * p1, * p2;
3. p1 = &x;
4. p2 = &y;
5. *p1 = 10;
6. *p2 = *p1;
7. p1 = p2;
```

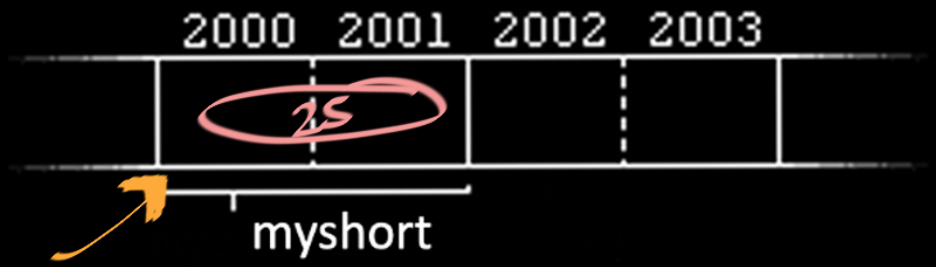


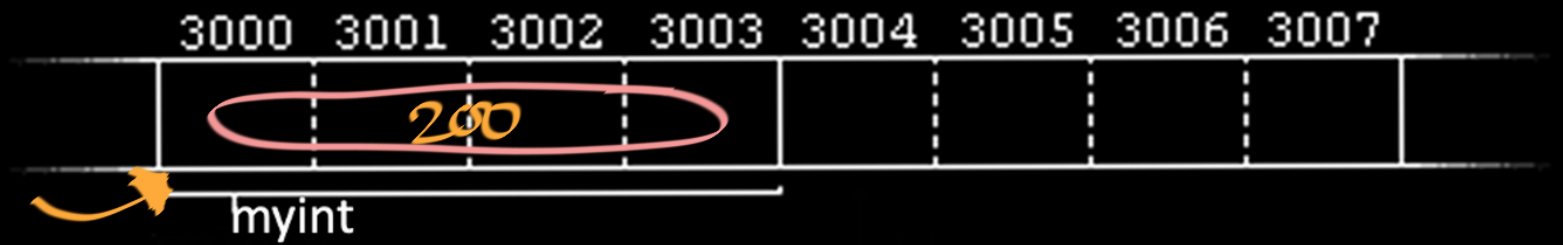
What is the output of the following code snippet?

```
1. int x = 5, y = 15;
2. int * p1, * p2;
3. p1 = &x;
4. p2 = &y;
5. *p1 = 10;
6. *p2 = *p1;
7. p1 = p2;
8. *p1 = 20;
9. printf("%d %d",x,y);
```







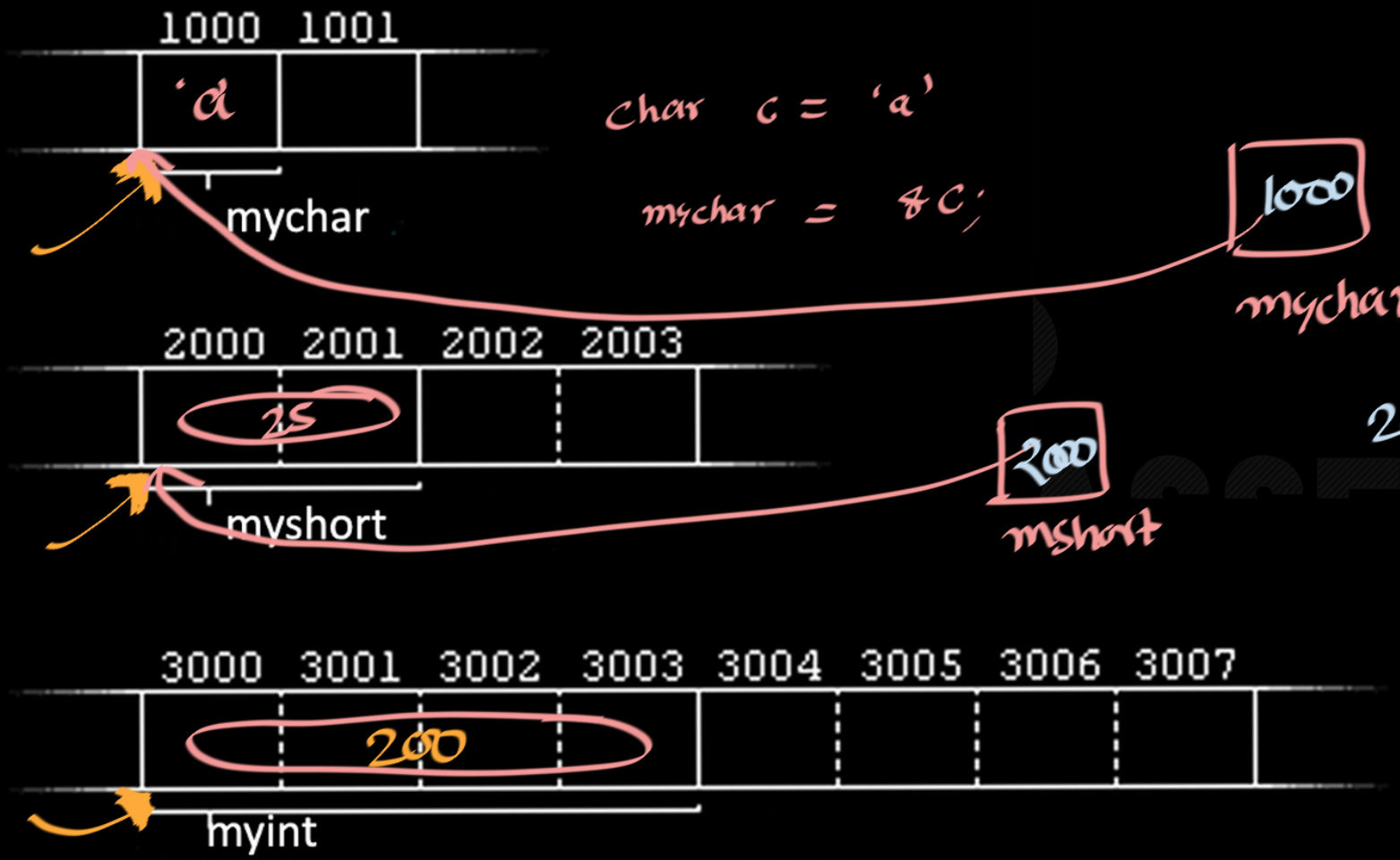



```
char *mychar;
short *myshort;
int *myint;
```

```
char c = 'a';
mychar = &c;
```

```
short s = 25;
myshort = &s;
int i = 200;
myint = &i
```

```
char *mychar;
short *myshort;
int *myint;
```



char c = 'a'

mychar = &c;

1000
mychar

(1000
Pf(mychar)

2000 ← Pf(myshort)

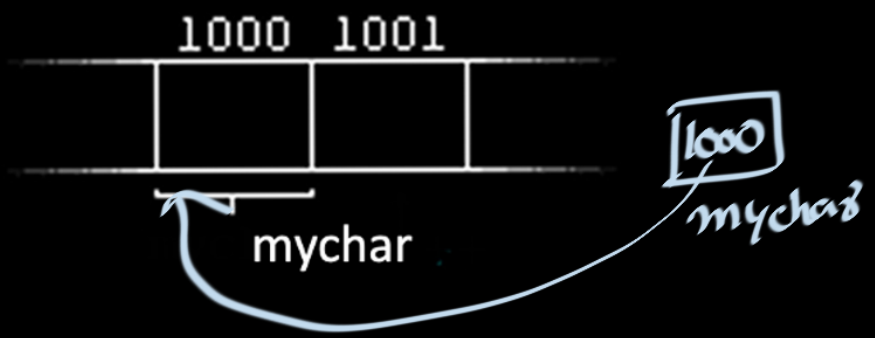
1000
myshort

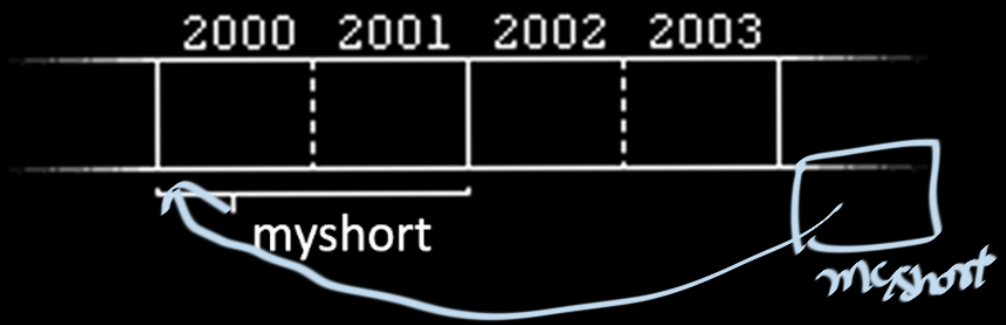
short s = 25;

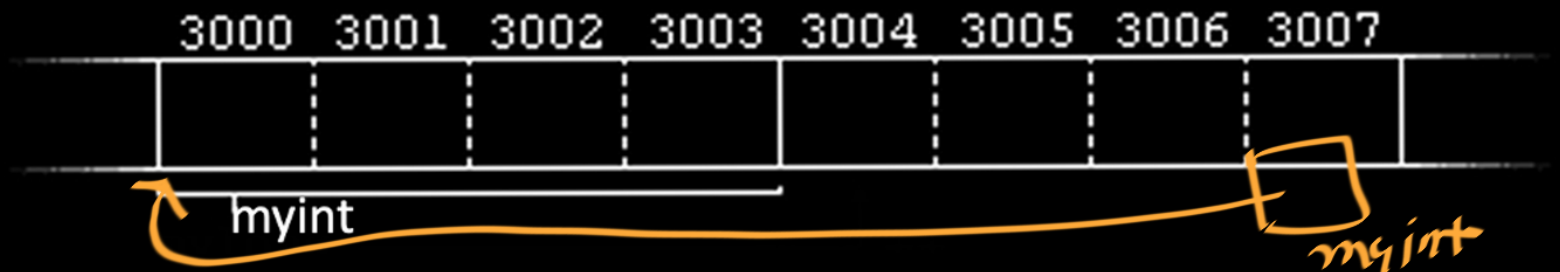
myshort = &s;

int i = 200;

myint = &i







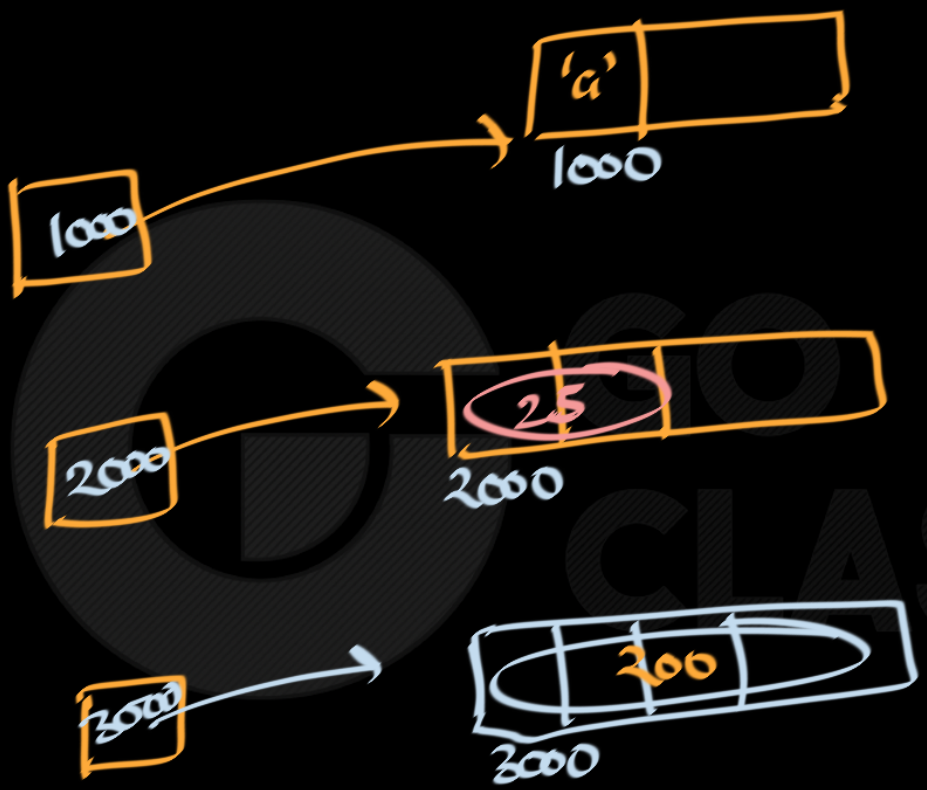

```
char *mychar;
short *myshort;
int *myint;
```

```
printf("value of mychar = %u", mychar);
printf("value of myshort = %u", myshort);
printf("value of myint = %u", myint);
```

1000

3000

2000


'a'
1000
1000
2000
25
2000
3000
200
3000

Passing pointer to function

```
main()
{
        int x;
        x = 20;
          fun( x);
        printf("%d\n",x);
}
  fun(int a)
{
    a = a+1;
}
```

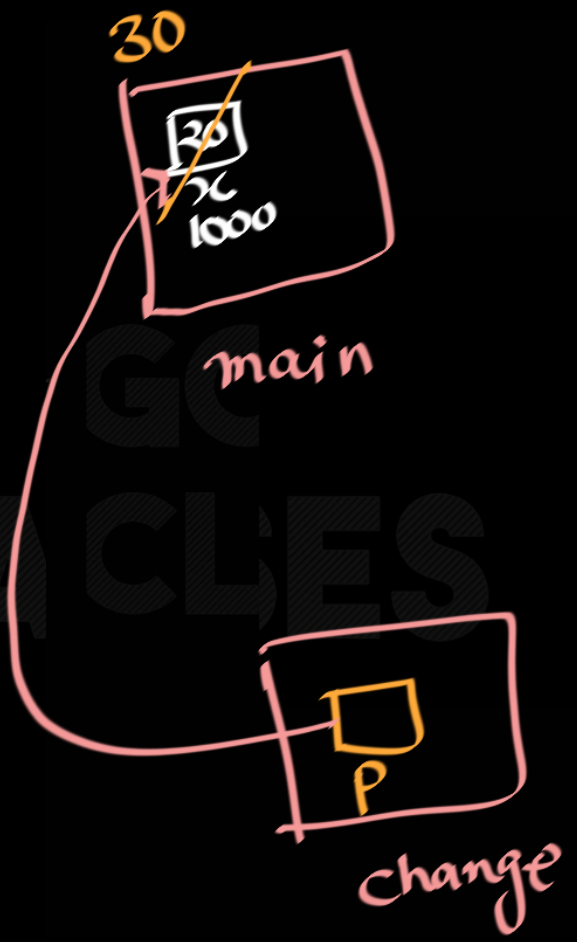
→ 20

```
main()
{
        int x;
        x = 20;
        change(&x);
        printf("%d\n",x);
}
change(int *p)
{
        *p = *p + 10;
}
```

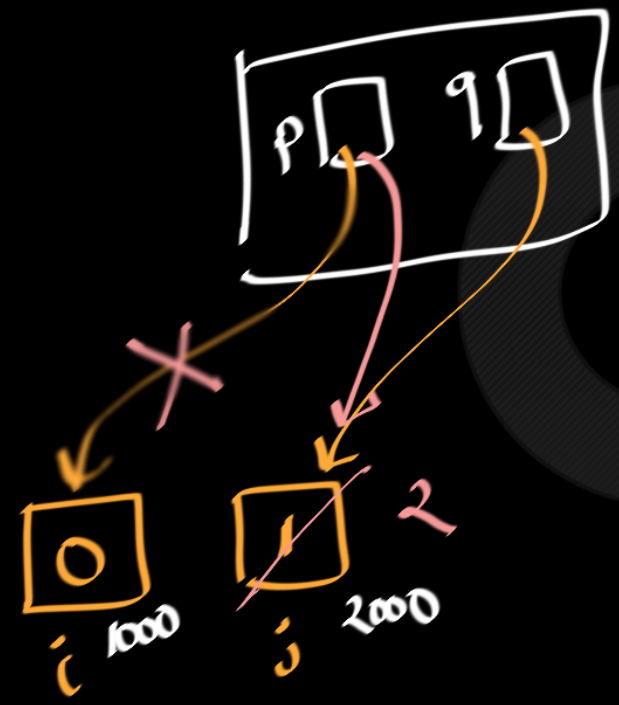
change (1000)

30

30



GATE 2010

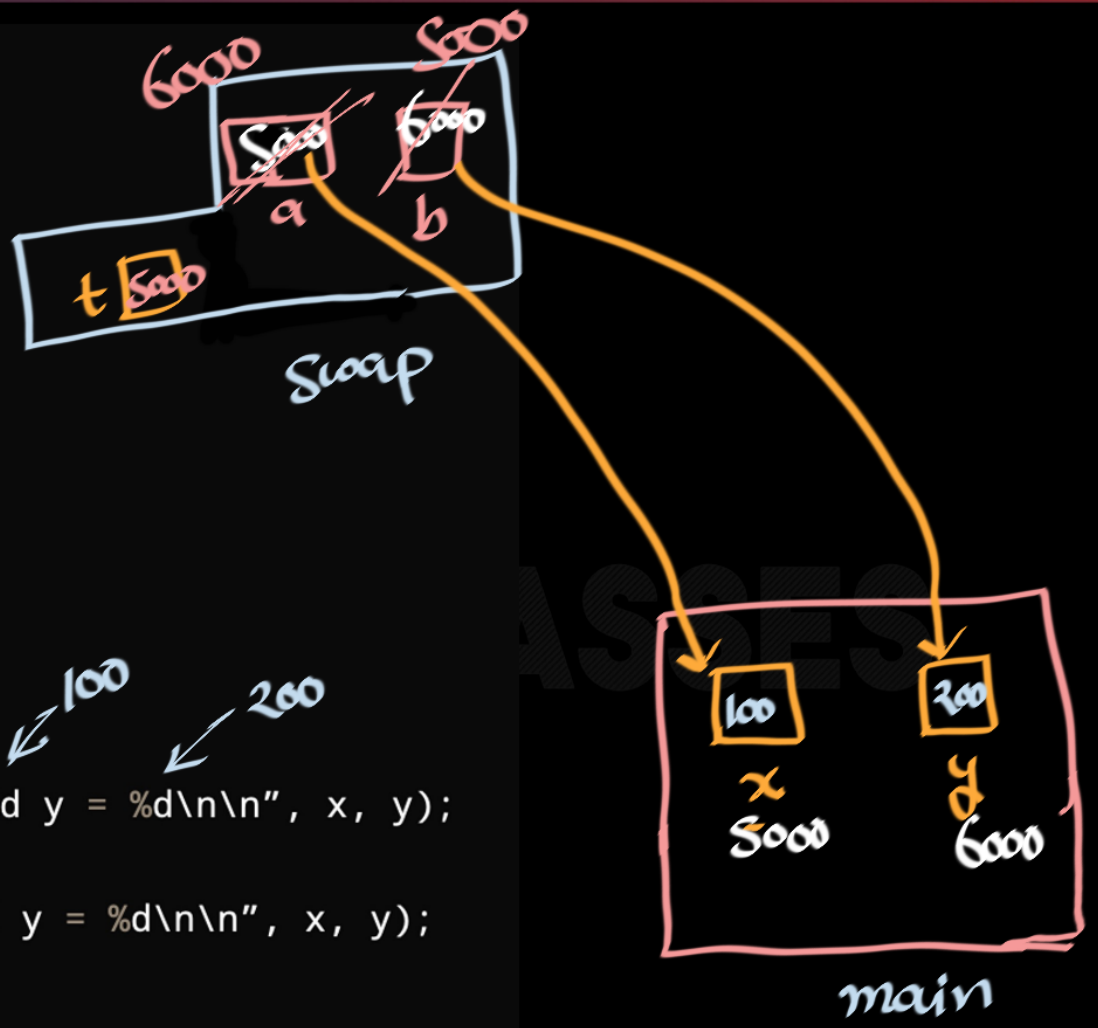



```c
#include<stdio.h>

void f(int *p, int *q) {
    p=q;
    *p=2;
}

int i=0, j=1;

int main() {
    f(&i, &j);
    printf("%d %d\n", i,j);
    return 0;
}
```

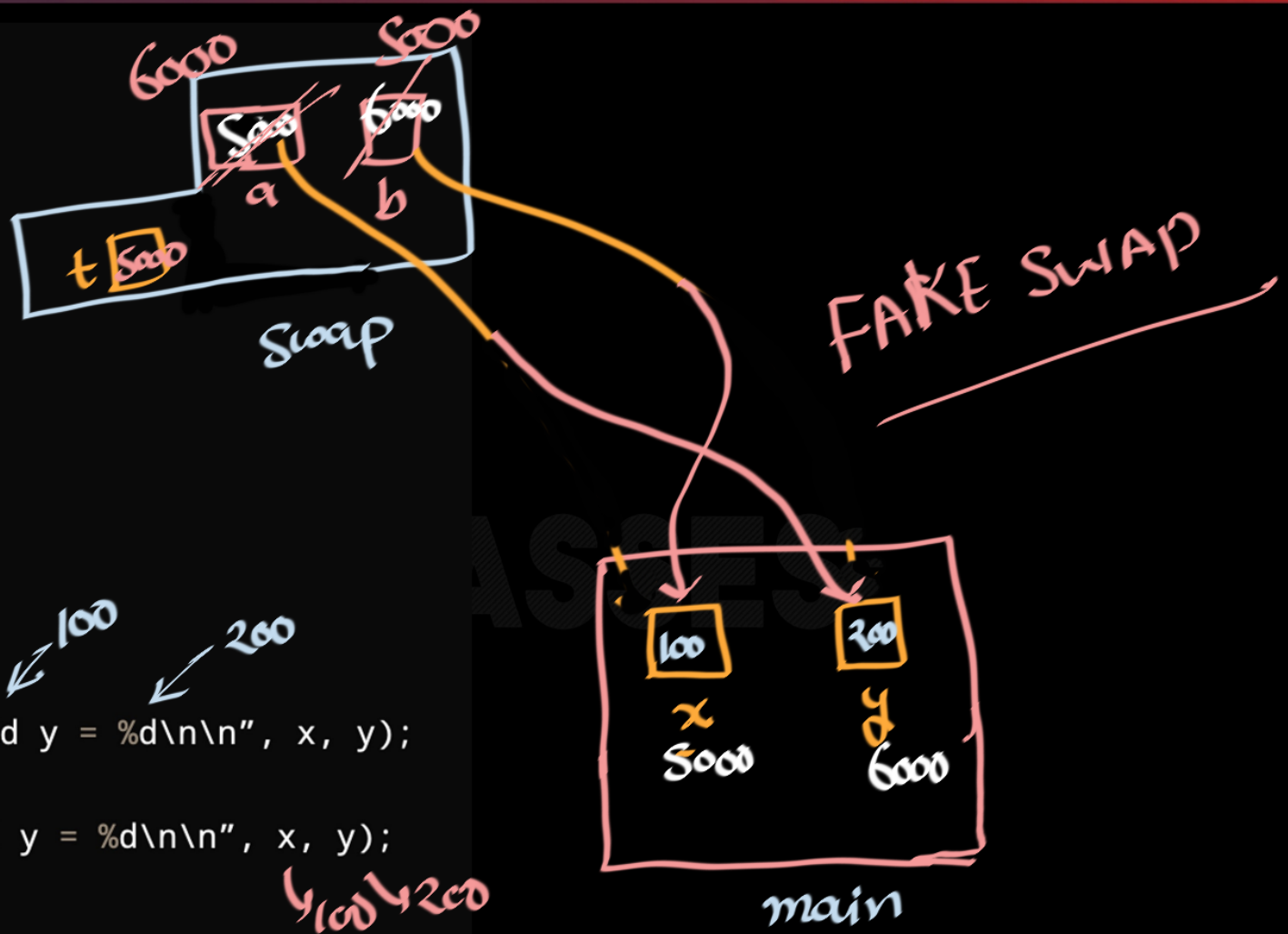
f(1000, 2000)

0 2

```c
swap (int *a, int *b)
{
    int *t;
    t = a;
    a = b;
    b = t;
}

main()
{
    int x, y;
    x = 100;
    y = 200;
    printf("Before call : x = %d y = %d\n\n", x, y);
    swap(&x,&y); /* call */
    printf("After call : x = %d y = %d\n\n", x, y);
}
```



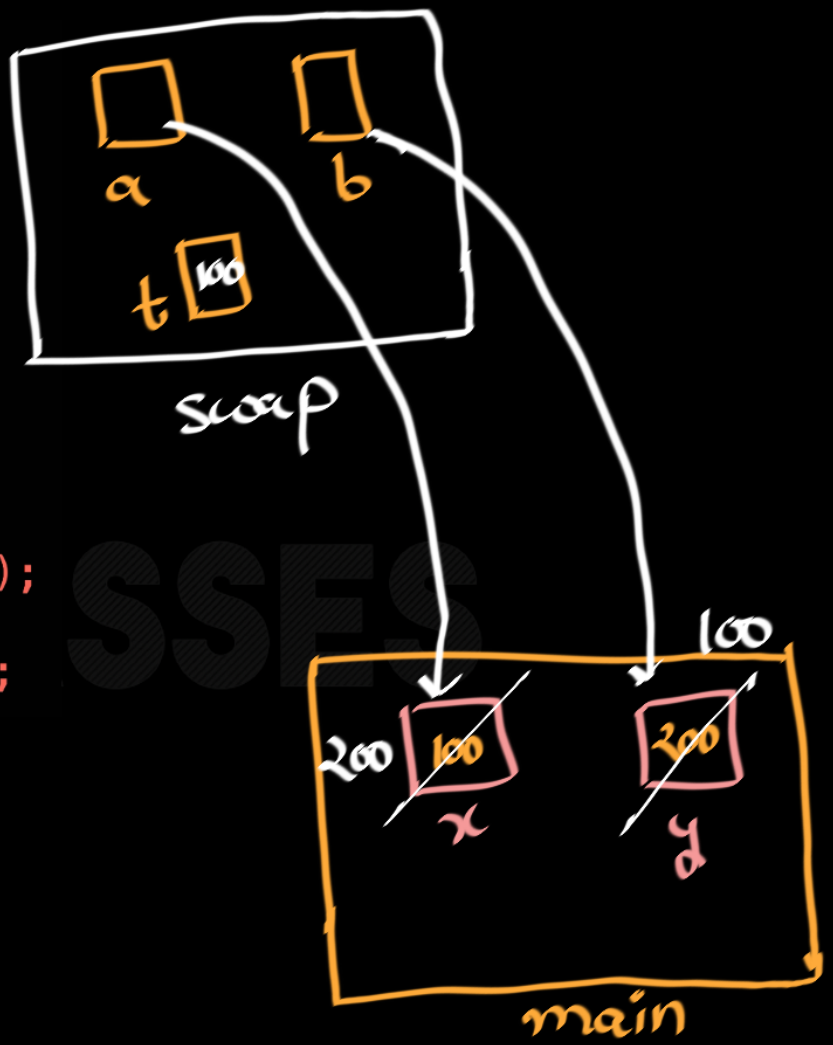
```c
swap (int *a, int *b)
{
    int *t;
    t = a;
    a = b;
    b = t;
}

main()
{
    int x, y;
    x = 100;
    y = 200;
    printf("Before call : x = %d y = %d\n\n", x, y);
    swap(&x,&y); /* call */
    printf("After call : x = %d y = %d\n\n", x, y);
}
```



```c
swap (int *a, int *b) {
    int t;
    t = *a;
    *a = *b;
    *b = t;
}

main() {
    int x, y;
    x = 100;
    y = 200;
    printf("Before call: x = %d y = %d\n\n", x, y);
    swap(&x, &y); /* call */
    printf("After call: x = %d y = %d\n\n", x, y);
}
```

100 200

200 100



```c
#include <stdio.h>
void swap (int *x, int *y)
{
    static int *temp;
    temp = x;
    x = y;
    y = temp;
}
void printab ()
{
    static int i, a = -3, b = -6;
    i = 0;
    while (i <= 4)
    {
        if ((i++)%2 == 1) continue;
        a = a + i;
        b = b + i;
    }
    swap (&a, &b);


    printf("a =  %d, b = %d\n", a, b);
}
main()
{
    printab();
    printab();
}
```

GATE 2006

A. $a = 0, b = 3$
$a = 0, b = 3$

B. $a = 3, b = 0$
$a = 12, b = 9$

C. $a = 3, b = 6$
$a = 3, b = 6$

D. $a = 6, b = 3$
$a = 15, b = 12$

```c
#include <stdio.h>
void swap (int *x, int *y)
{
    static int *temp;
    temp = x;
    x = y;
    y = temp;
}

main()
{
    printab();
    printab();
}
```

```c
void printab ()
{
    static int i, a = -3, b = -6;
    i = 0;
    while (i <= 4)
    {
        if ((i++)%2 == 1) continue;
        a = a + i;
        b = b + i;
    }
    swap (&a, &b);

    printf("a =  %d, b = %d\n", a, b);
}
```

i=0

i=1

a=6 b=3

i=2

i=3

a=15 b=12

i=4

```c
#include <stdio.h>
void swap (int *x, int *y)
{
    static int *temp;
    temp = x;
    x = y;
    y = temp;
}
void printab ()
{
    static int i, a = -3, b = -6;
    i = 0;
    while (i <= 4)
    {
        if ((i++)%2 == 1) continue;
        a = a + i;
        b = b + i;
    }
    swap (&a, &b);


    printf("a =  %d, b = %d\n", a, b);
}
main()
{
    printab();
    printab();
}
```

GATE 2006

A. $a = 0, b = 3$
$a = 0, b = 3$

B. $a = 3, b = 0$
$a = 12, b = 9$

C. $a = 3, b = 6$
$a = 3, b = 6$

D. $a = 6, b = 3$
$a = 15, b = 12$

Answer is D

```c
#include<stdio.h>
#define print(x) printf("%d", x)

int x;
void Q(int z)
{
        z+=x;
        print(z);
}

void P(int *y)
{
        int x = *y + 2;
        Q(x);
        *y = x - 1;
        print(x);
}
main(void) {
        x = 5;
        P(&x);
        print(x);
}
```

GATE 2003

The output of this program is:

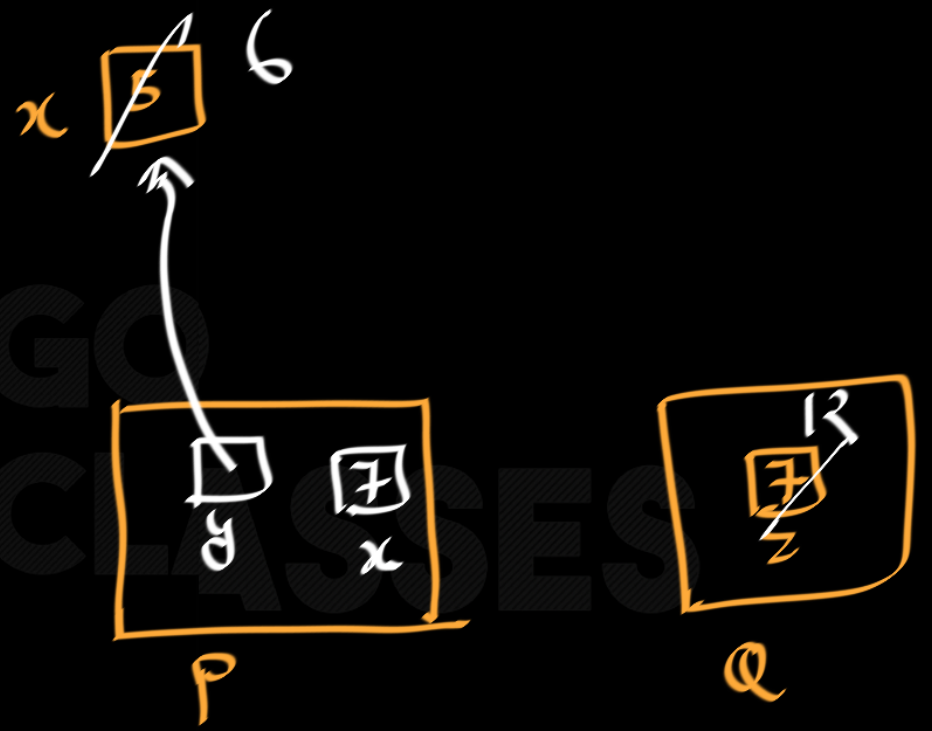
A. 12 7 6

B. 22 12 11

C. 14 6 6

D. 7 6 6

```c
#include<stdio.h>
#define print(x) printf("%d", x)

int x;
void Q(int z)
{
        z+=x;
        print(z);
}

void P(int *y)
{
        int x = *y + 2;
        Q(x);
        *y = x - 1;
        print(x);
}
main(void) {
        x = 5;
        P(&x);
        print(x);
}
```



13 7 6

GO CLASSES

Array

Array

- Array is a data structure which can represent a collection of data items which have the same data type (float/int/char/...)

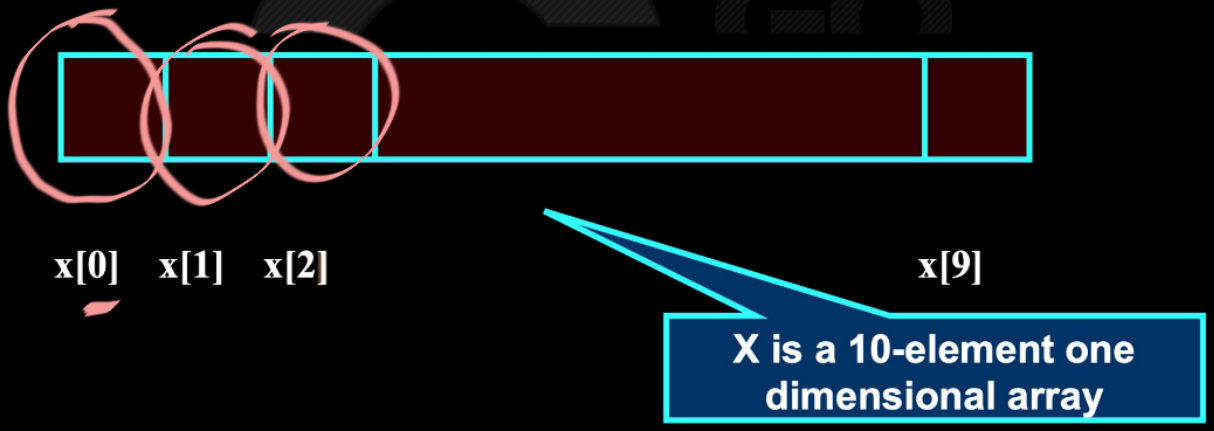
- All the data items constituting the group share the same name

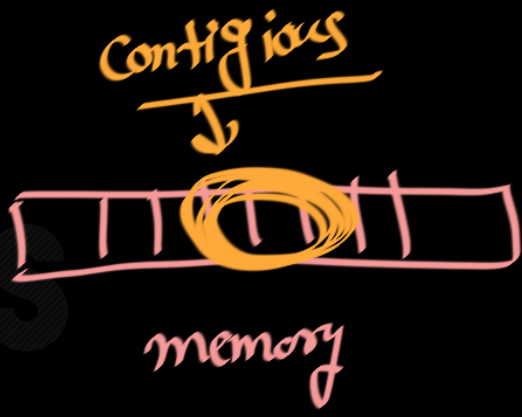
```
int x[10];
```

You have a collection of 10 integers.

Array (Contd..)

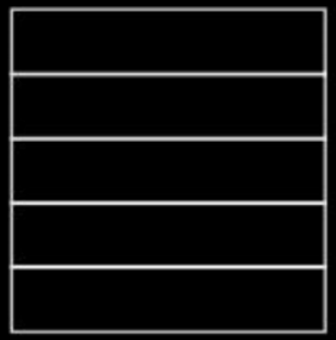
- Individual elements are accessed by specifying the index





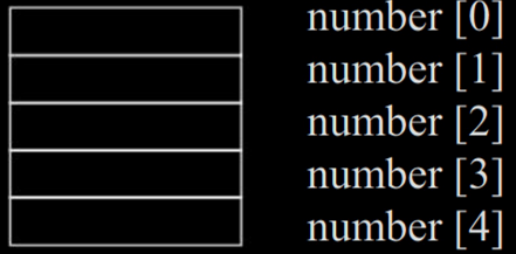
An Example

```
int number[5];
```

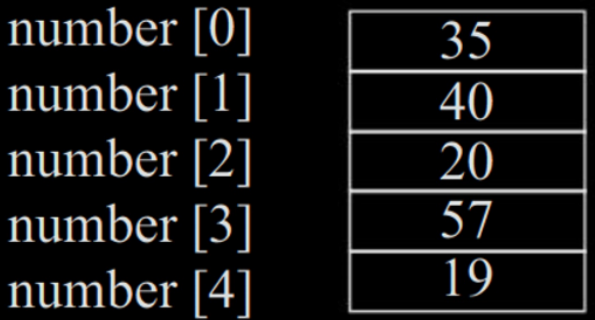


An Example

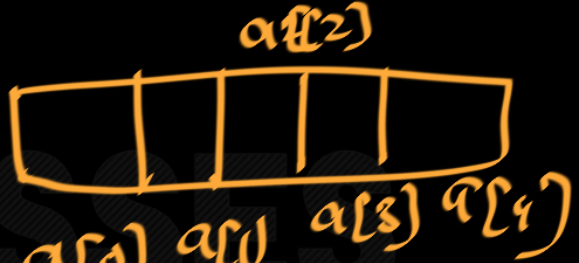
```
int number[5];
```




```
number[0] = 35;
number[1] = 40;
number[2] = 20;
number[3] = 57;
number[4] = 19;
```



```
main ( )
{
    int a[5];
    a[0] = 10
    Pf ("%d", a[0])
}
```



a[0] → 10

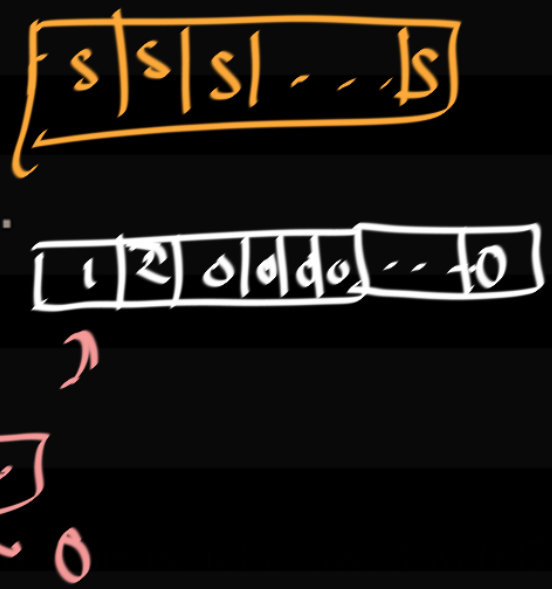
Array Initialization

1.
```
int myArray[10] = { 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 };
```

2.
```
int myArray[10] = { 1, 2 }; // initialize to 1,2,0,0,0...
```

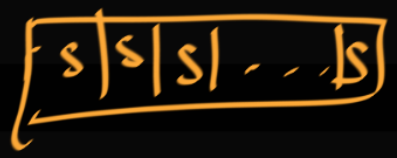
3.
```
int myArray[10] = { 0 }; // all elements 0
```

4.
```
static int myArray[10]; // all elements 0
```



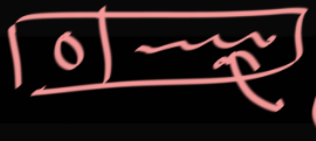
Array Initialization

1. `int myArray[10] = { 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 };`



2. `int myArray[10] = { 1, 2 }; // initialize to 1,2,0,0,0...`

3. `int myArray[10] = { 0 }; // all elements 0`



4. `static int myArray[10]; // all elements 0`

5. `int myArray[10];` ← local — all values are garbage

≈ global — " " — zero

initialise in a such a way s.t. (10)

3rd and 5th elements are 1, 2 respectively

a[2] a[4]

and everything else is zero.

a[2] a[4]

int a[10] = {0, 0, 1, 0, 2}

GO CLASSES

```
int a[10] = {0};
```

Array Initialization (Contd..)

In case of character array default value is null character '\0' (ASCII Value Zero)

```
1. char t[5] = {'a', 'b', 'c', 'd'};  // initialize to 'a','b','c','d','\0'

2. char t[4] = {'a', 'b', 'c', 'd'};  // initialize to 'a','b','c','d'


3. char s[5] = "abcd"; // initialize to 'a','b','c','d','\0'

4. char s[4] = "abcd"; // initialize to 'a','b','c','d'
```

(Short form of 1st)

(Short form of 2nd)

GO CLASSES

char t[7] = "GOclass"

t[6] ↓ t[0]

|||

char t[7] = {'G', 'O', 'c', 'l', 'a', 's', 's'}

GO CLASSES

```
char s[5] = {'a', 'b', 'c'};
```

remaining 2 is null

!!!

```
char s[5] = "abc";
```

If dimension is not specified

The compiler will deduce the dimension from the initializer list

```
int myArray[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

```
char s[] = "abcd";  // equivalent to char s[5] = "abcd";
```

GO CLASSES

char c[ ] = "abcd"

↗ size of c is 5 char.

GO CLASSES

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

Size of c is 4 characters.

GO CLASSES

char c[ ] = "abcd"

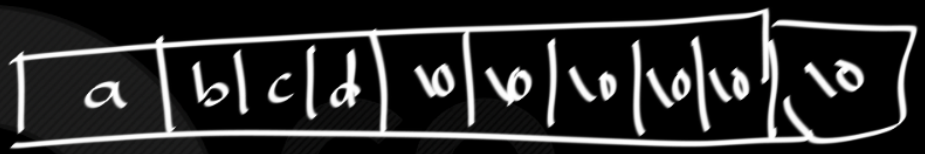
size of c is 5 char.

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

size of c is 4 characters

GO CLASSES

char c[10] = "abcd"



char c[4] = "abcd"

char c[5] = "abcd"

GO CLASSES

char c[ ] = "abcd";

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