

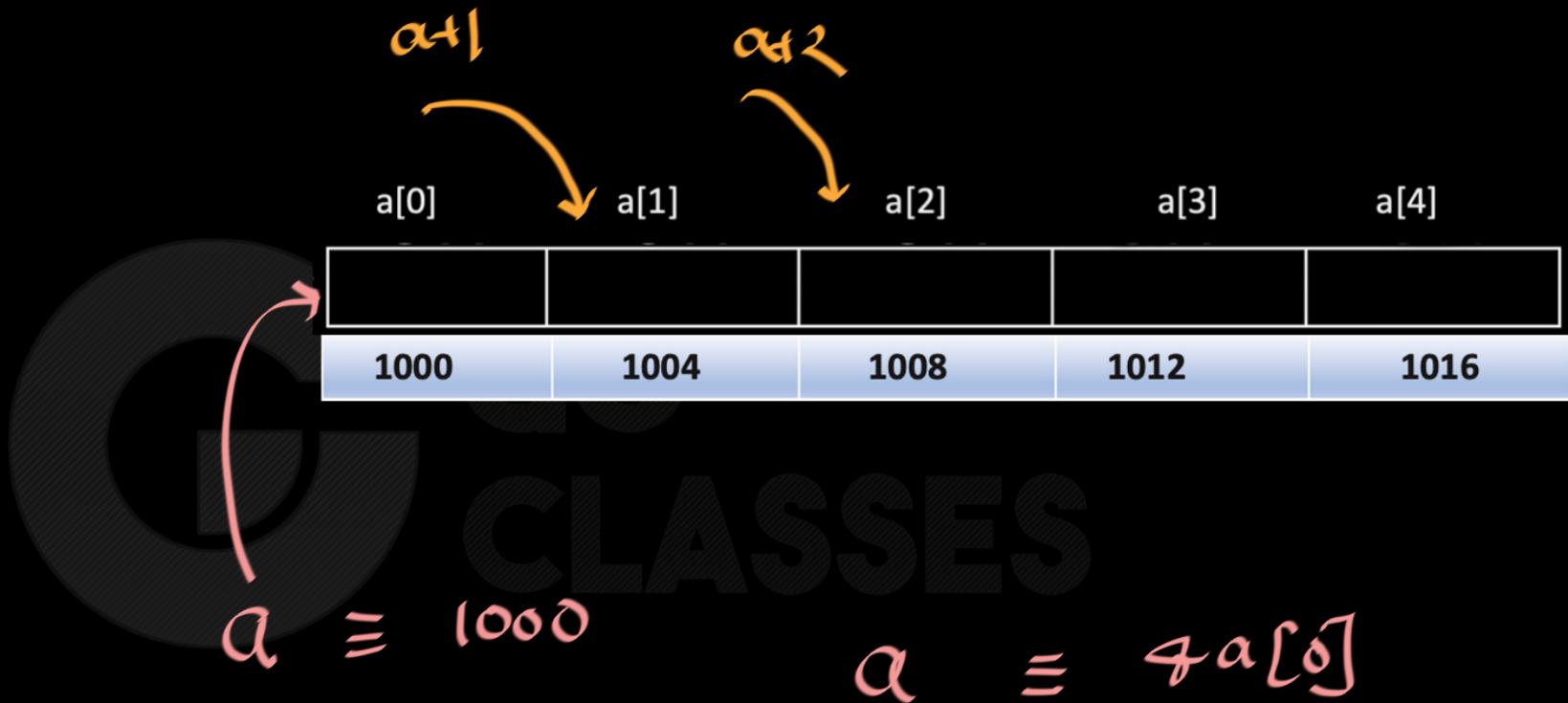


# Arrays and Pointers



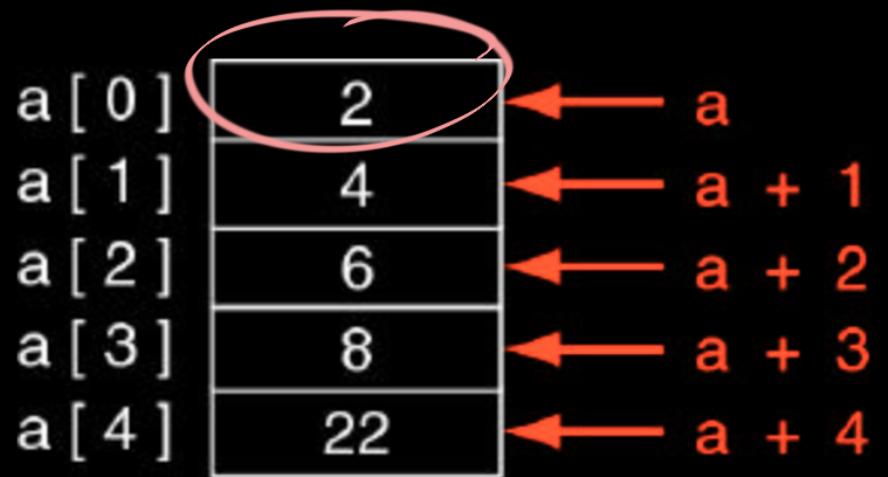
# C Programming

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





# C Programming



$$a \equiv *a[0]$$

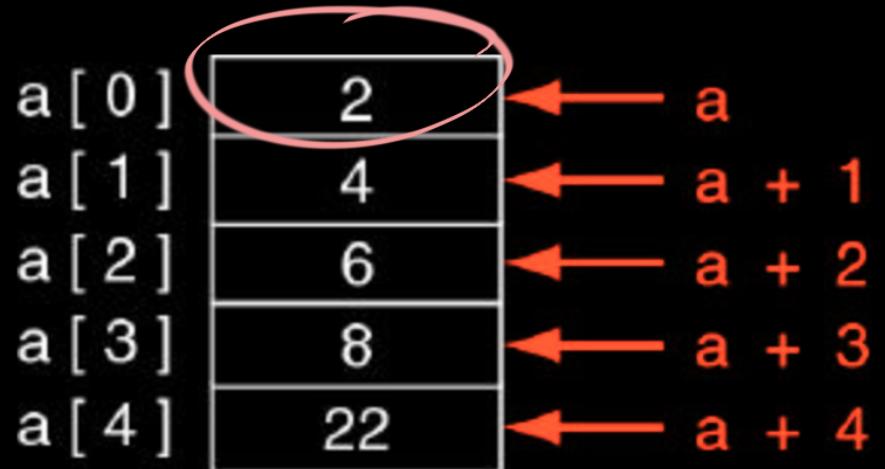
$$a+1 \equiv *a[1]$$

$$a+2 \equiv *a[2]$$

$a[0]$        $*a+0$



# C Programming

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

$$a[2] \equiv *(*(a+2))$$
$$a[0] \equiv *(*(a+0))$$



a[i] -

- a represents address of first element in array
- a+0 represents address of first element in array
- a+1 represents address of second element in array
- a[i] is same as \*(a+i)

$\underbrace{\ast(a+i)}$   
a[i]



## Question:

If `a` is an array of integers, then `a[ i ]` is equivalent to

- A. `*a+i`
- B. `a+i`
- C. `*(a+i)`
- D. none of the above

  
 $\brace{ } \downarrow$   
 $*(a+i)$



## Question:

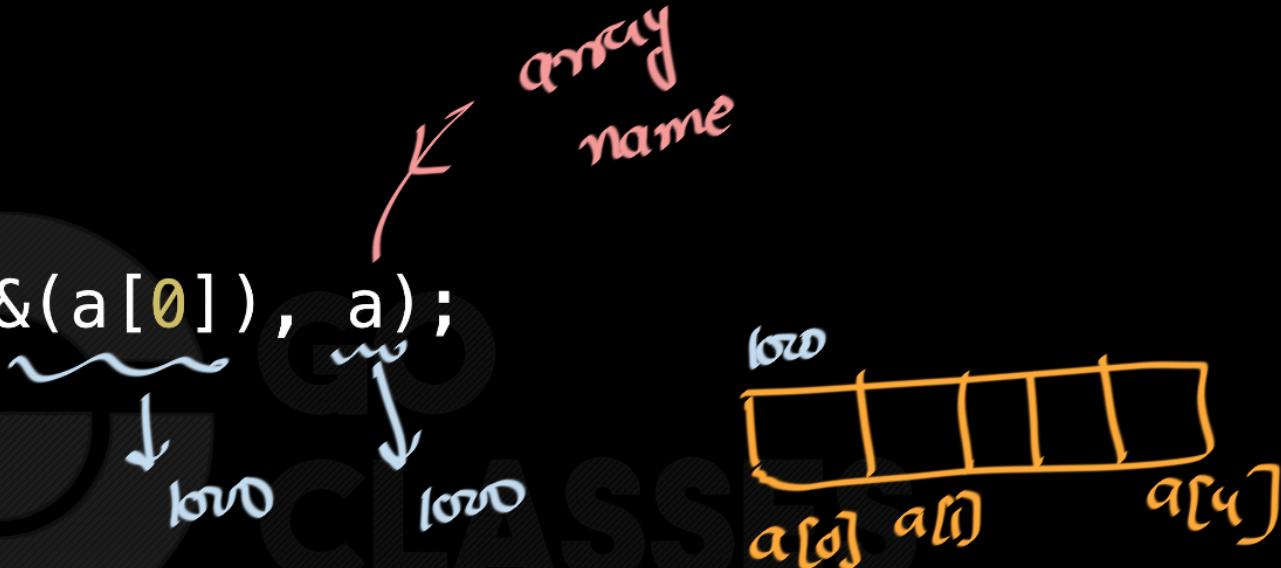
If  $a$  is an array of integers, then  $a[i]$  is equivalent to

- A.  $*a+i$
- B.  $a+i$
- C.  $*(a+i)$
- D. none of the above



$*(a+i)$

```
int a[5];  
printf("%p %p", &(a[0]), a);
```



The two values printed are identical

Small  
detail

Suppose

address is 64 bits  
integer is 32 bits

%P      f

designated

formed specifier  
to print addresses.

%u

this is for  
unsigned integers.

address

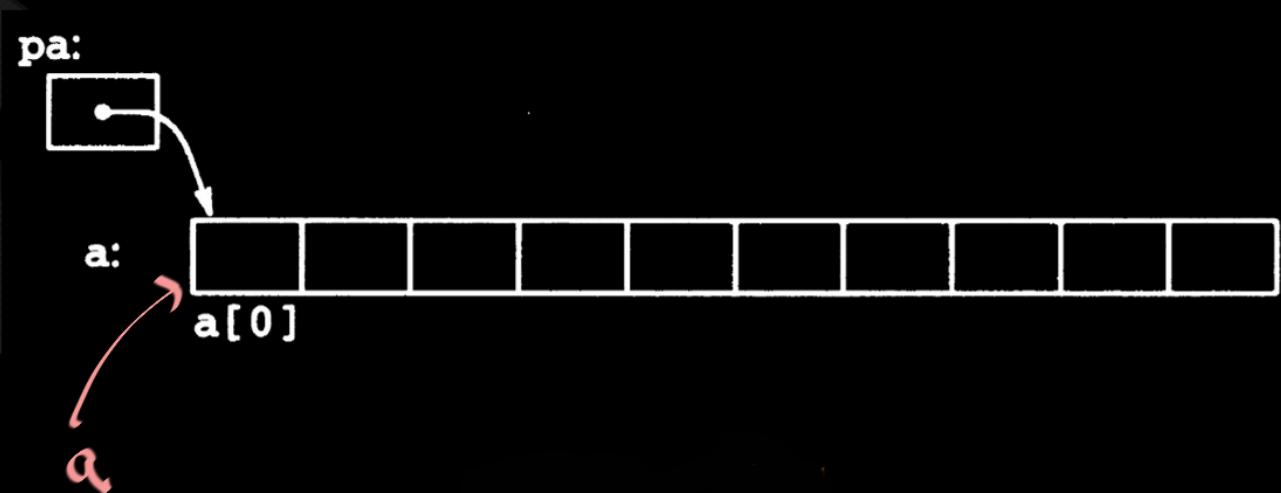
$\{ \overline{110010 \dots 00} \} \overline{00101000}$   
32 bits  
↓  
%P      %u

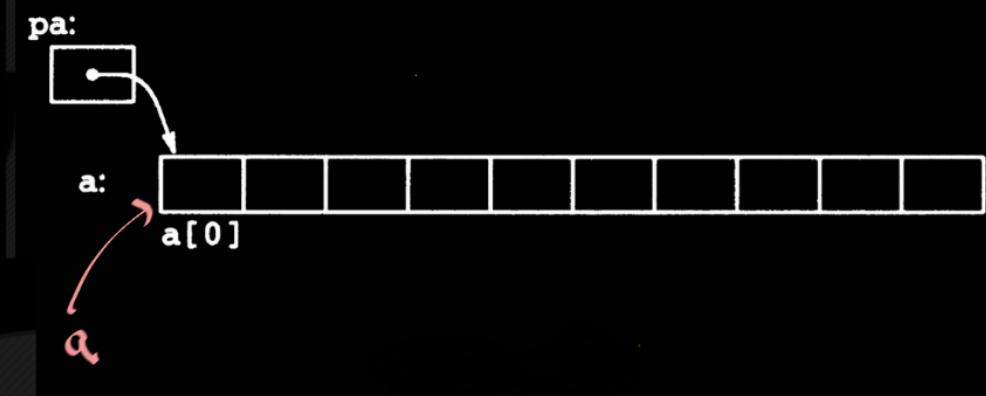


# C Programming

```
int *pa;  
  
int a[10];  
  
pa = &a[0];
```

pa ≡ a ≡ a[0]

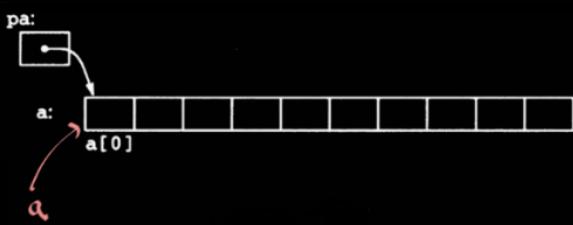




Suppose  $\text{pa}$  is also pointing to the first element of array.

What is the difference b/w  $\text{pa}$  and  $a$ ?

→ 'a' can not be used on left hand side  
 $a = a + 1$  (illegal)



Suppose  $\text{pa}$  is also pointing to the first element of array.

What is the difference b/w  $\text{pa}$  and  $a$ ?

→ 'a' can not be used on left hand side.  
 $a = a + 1$  (illegal)

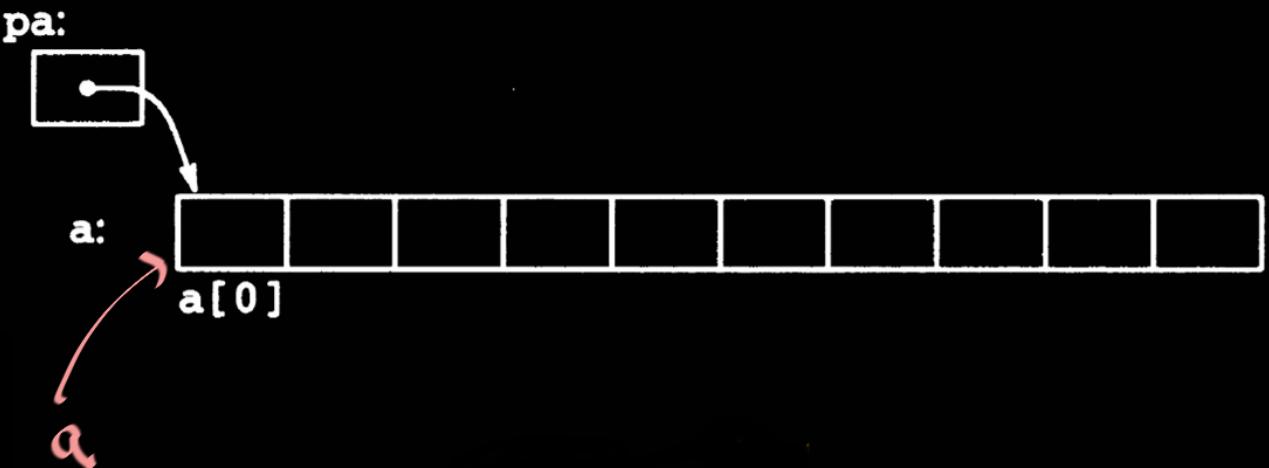
but we can use for  $\text{pa}$ .

→  $\text{pa}$  has its own box ,  $a$  doesn't have its own box

→  $\text{sizeof}(\text{pa}) \hookrightarrow$  Size of address (8 bytes)       $\text{sizeof}(a) \hookrightarrow 5 \times 4 \text{ bytes} = 20 \text{ bytes}$

No. of integers  $\times$  size of one integ

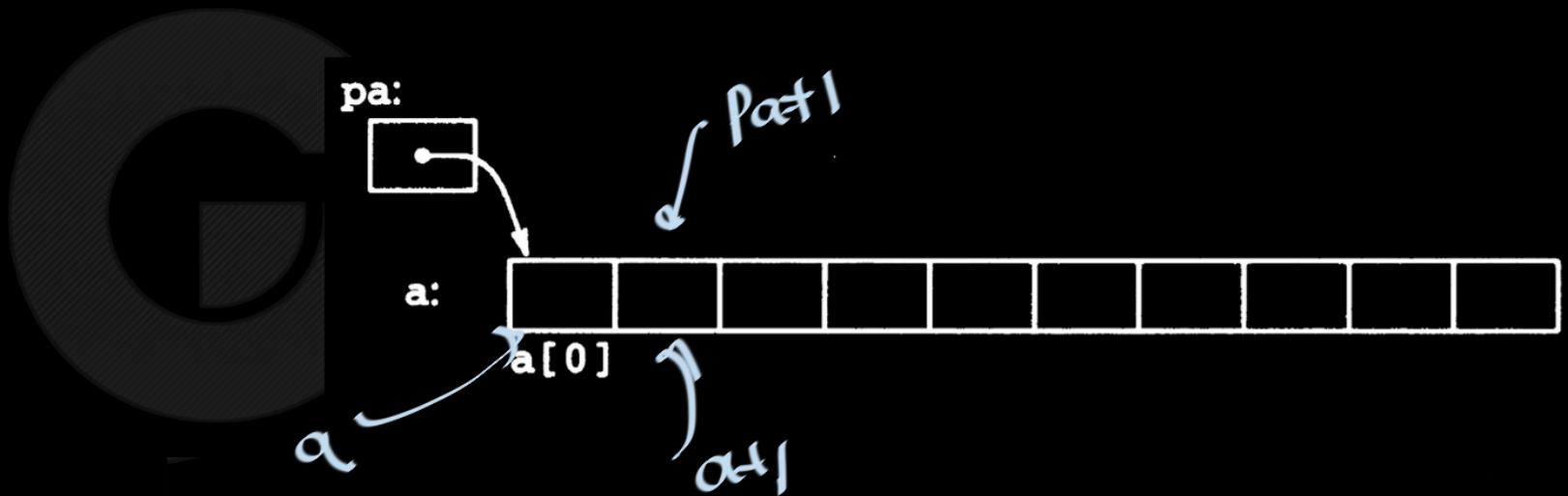
```
int *pa;  
  
int a[10];  
  
pa = &a[0];  
or  
pa = a;
```





# C Programming

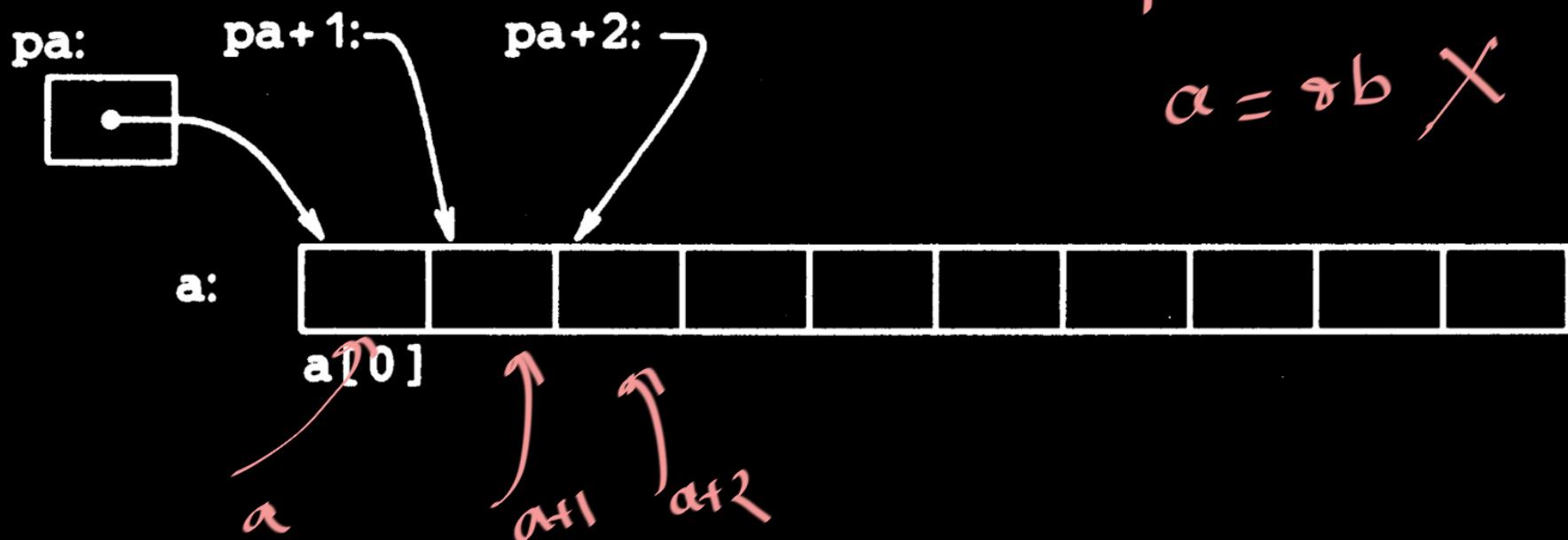
```
int *pa;  
  
int a[10];  
  
pa = &a[0];
```



**pa+1** points to ??



# C Programming



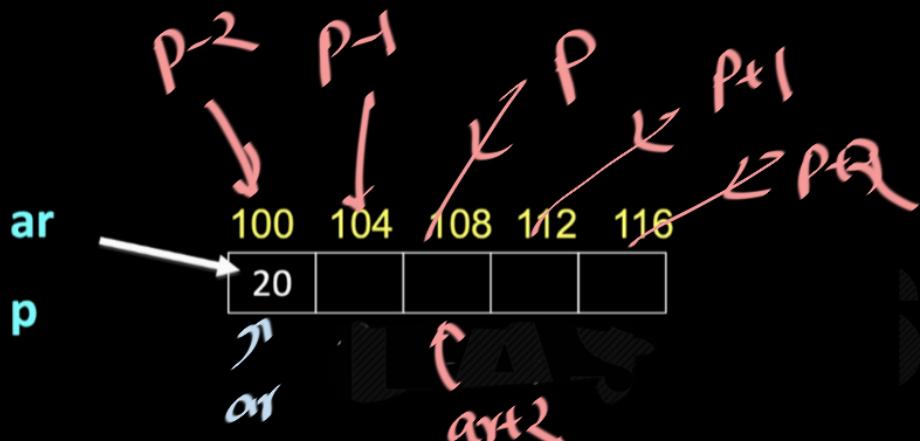
## Question 1:

Which of the assignment statements leads to compilation error ?

```
int *p, ar[5];
```

- i)  $p=ar+2;$
- ii)  $ar=p+1;$

- A.  $p=ar+2;$
- B.  $ar=p+1;$
- C. Both statements result in error at compile time
- D. Neither results in a compilation error



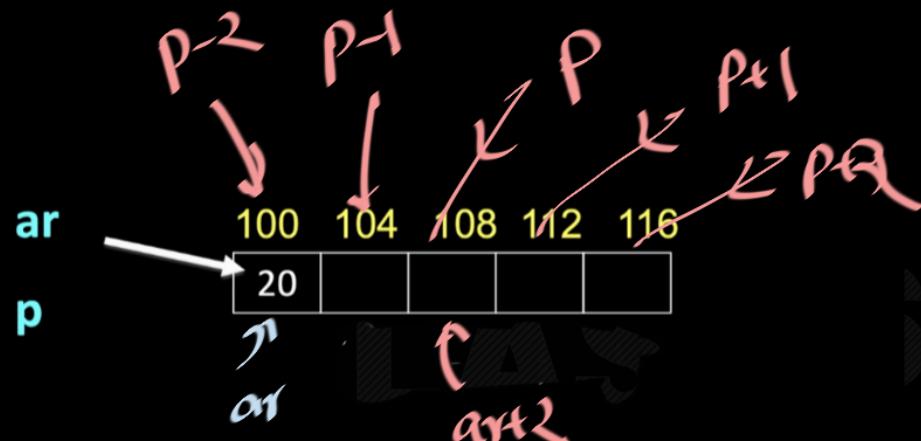
## Question 1:

Which of the assignment statements leads to compilation error ?

```
int *p, ar[5];
```

- i)  $p=ar+2;$
- ii)  $ar=p+1;$

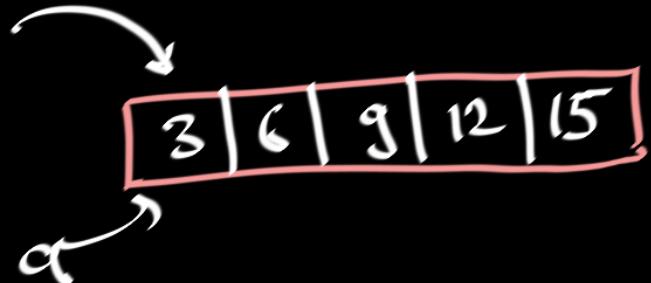
- A.  $p=ar+2;$
- B.  $ar=p+1;$
- C. Both statements result in error at compile time
- D. Neither results in a compilation error





## Question 2:

addr



```
int a[] = {3, 6, 9, 12, 15};  
int* addr = &(a[0]);
```

- What is the value of  $(\ast \text{addr}) + 4$ ?

---

7

- What is the value of  $\ast(\text{addr} + 4)$ ?

---

15

int\* addr = &(a[0]);

or

int\* addr = a;

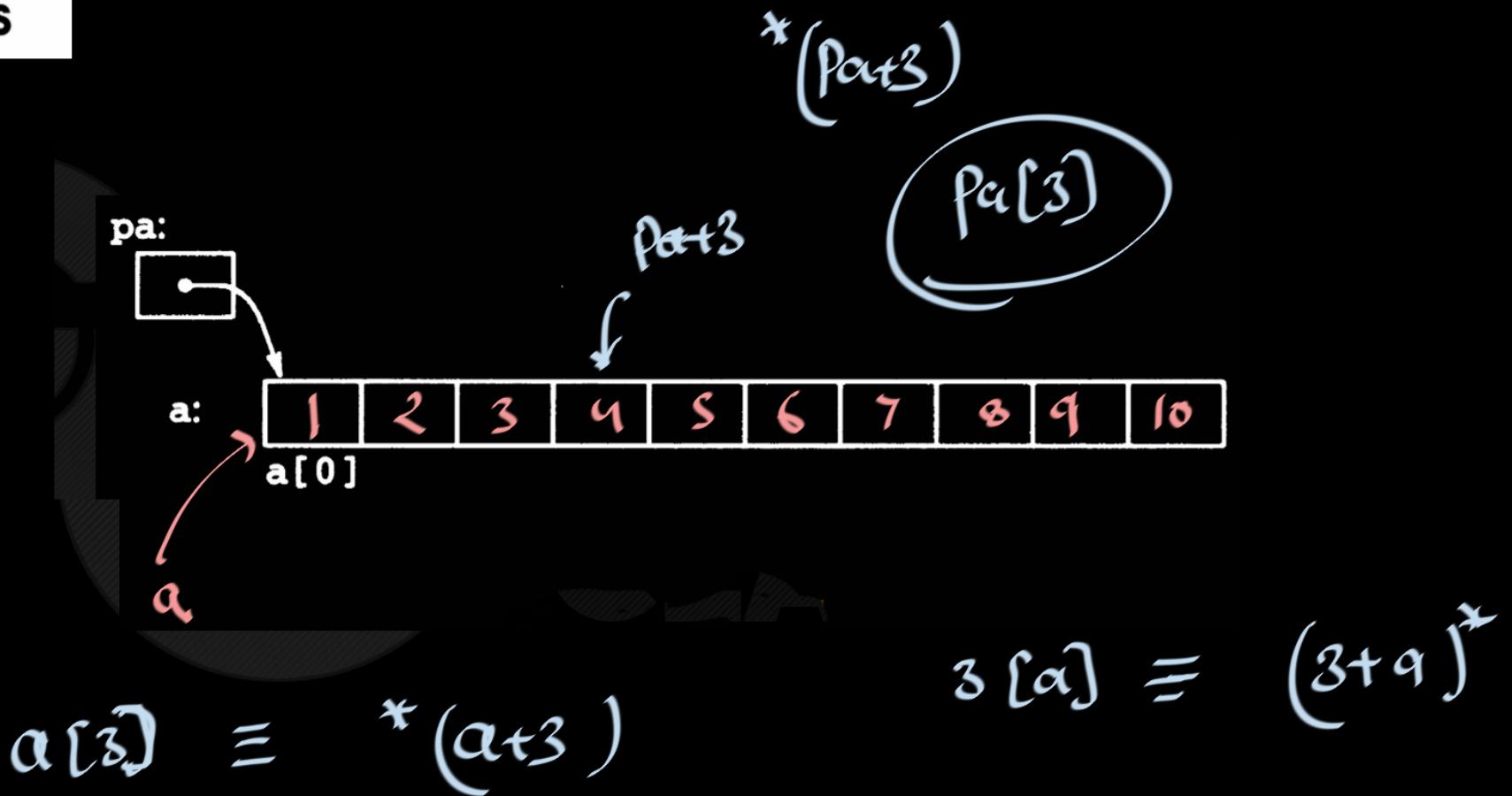
```
int* addr = &(a[0]);
```

int \*addr;

\*addr = &a[0];

int \*addr;

addr = &a[0];



pa:



a:



a

6 ways to access  $a[i]$

- 1)  $a[3]$
- 2)  $3[a]$
- 3)  $*(\&a + 3)$

- 4)  $\&a[3]$
- 5)  $3[\&a]$
- 6)  $*(\&a + 3)$

pa:



a:

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

a[0]

a

1) a[3]

a[3]



2) 3[a]

3[a]

3) \*(a+3)

4) \*pa[3]

3[pa]

5) \*(pa+3)

preferred  
ways



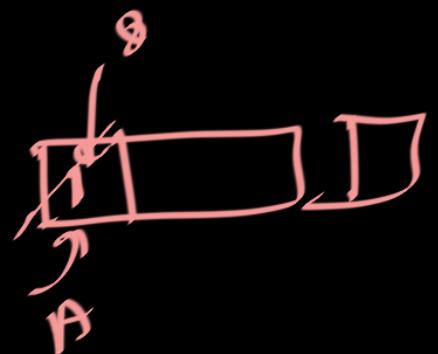
## Question 3:

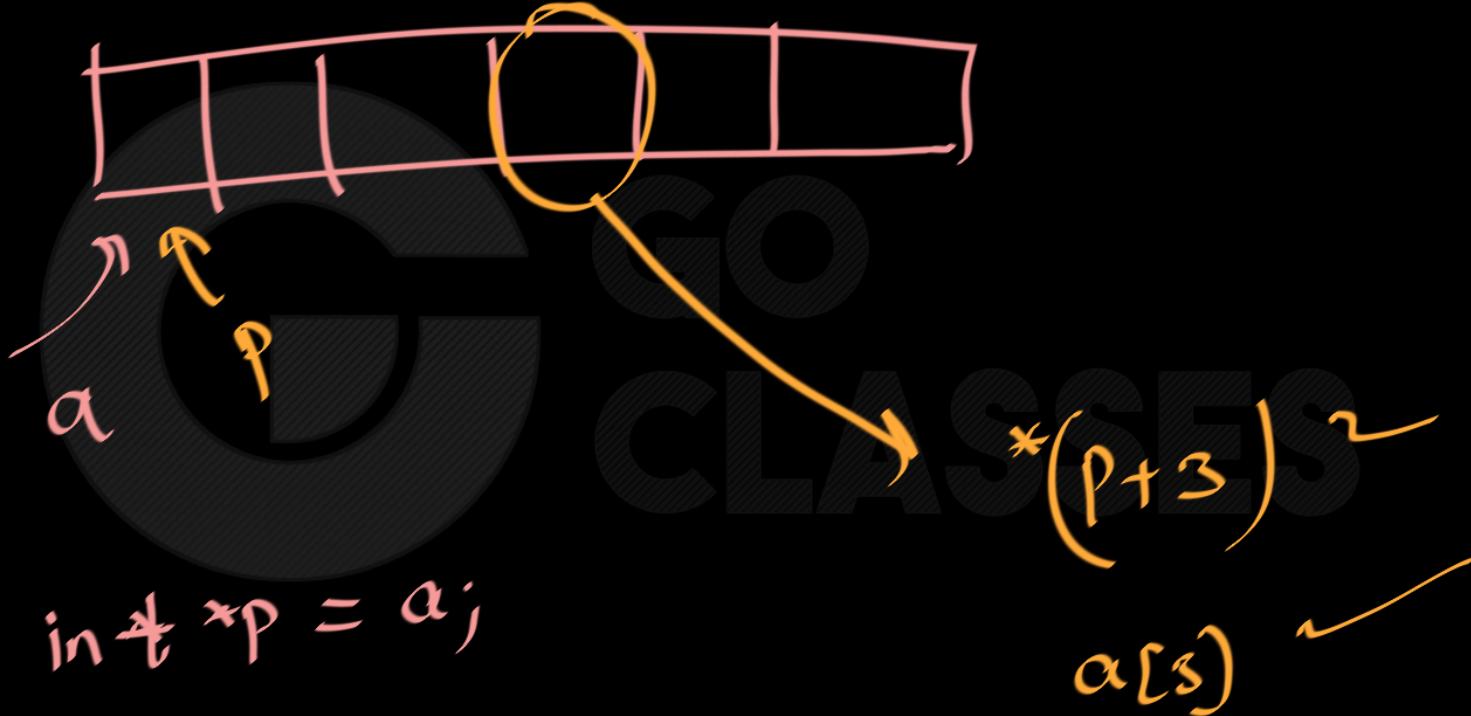
Given the declarations `int A[20], *p, x; p = A;`)

which of the following statements is NOT TRUE

- [a.]  $p = A$ ; is equivalent to  $p = \&A[0]$ ; T
- [b.]  $x = p[ 4 ]$ ; is equivalent to  $x = *(p + 4)$ ; T
- [c.]  $x = p[ 3 ]$ ; is equivalent to  $x = *(A + 3)$ ; T
- [d.]  $x = *A + 7$ ; is equivalent to  $x = A[ 7 ]$ ; F

$\downarrow *(\text{A}+7)$





$$\text{int } *p = \alpha;$$



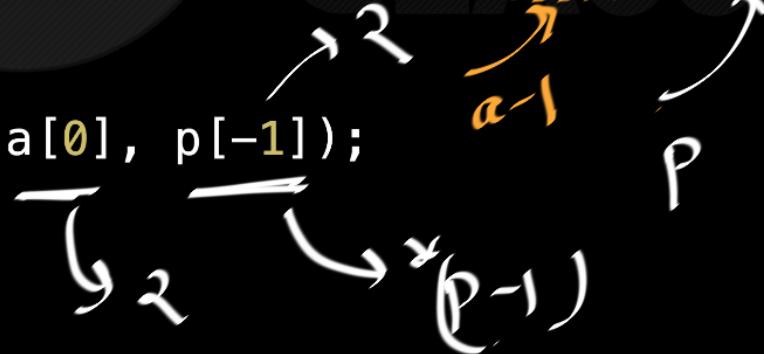
## Question 4:

```
#include <stdio.h>

int main(void) {
    int a[5] = {2, 4, 6, 8, 22};
    int *p;
    p = &(a[1]);
    printf("%d %d\n", a[0], p[-1]);
    return 0;
}
```

\*( $a - 1$ )

2	4	6	8	22
---	---	---	---	----



```
#include <stdio.h>
int main(void) {
    int a[5] = {2, 4, 6, 8, 22};
    int *p;
    p = &(a[1]);
    printf("%d %d\n", a[0], p[-1]);
    printf("%d\n", p[0]);
    return 0;
}
```

Desktop - zsh - 63x10

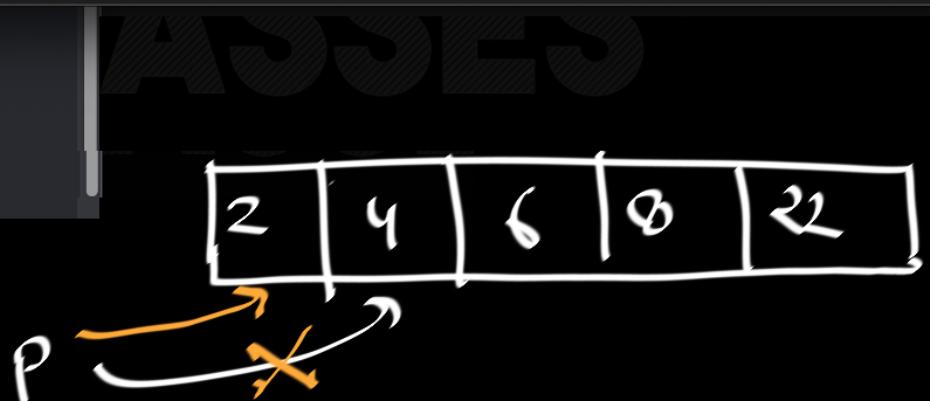
```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
2 2
4
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

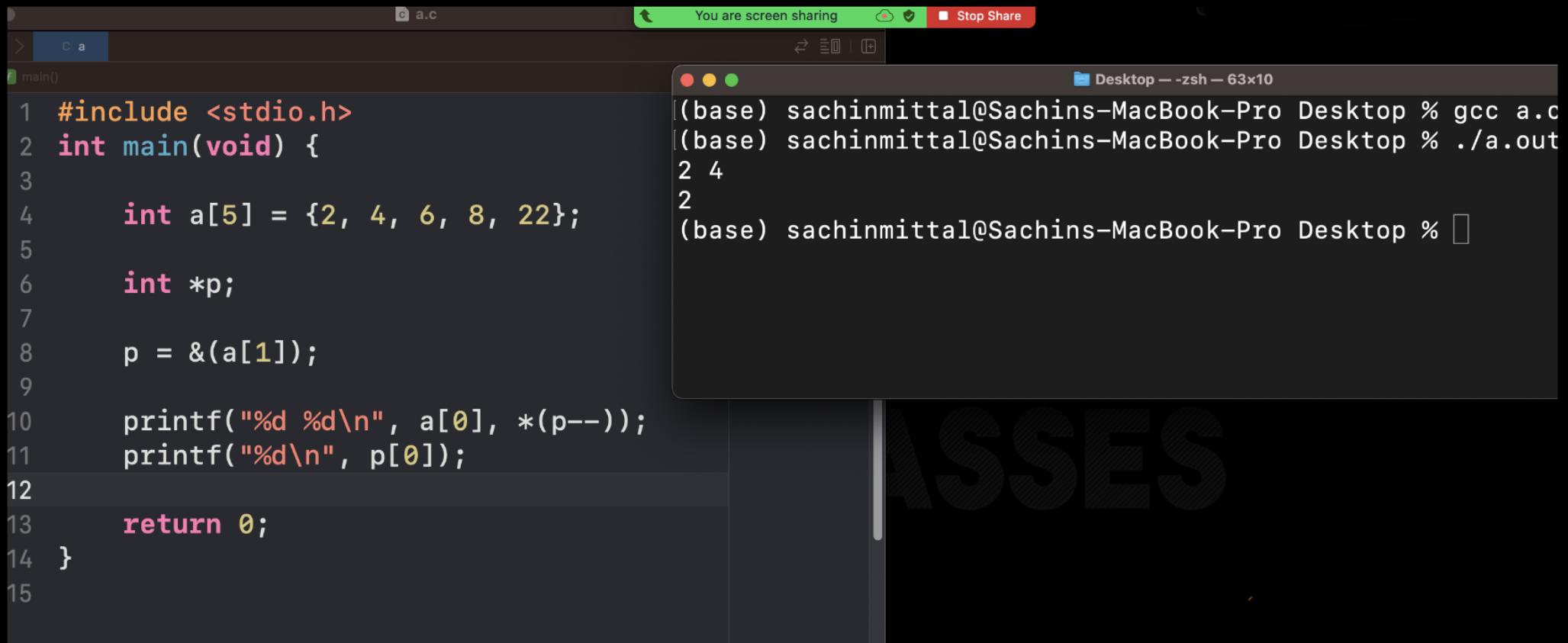


P

```
#include <stdio.h>
int main(void) {
    int a[5] = {2, 4, 6, 8, 22};
    int *p;
    p = &(a[1]);
    printf("%d %d\n", a[0], *(--p));
    printf("%d\n", p[0]);
    return 0;
}
```

Desktop — zsh — 63x10  
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c  
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out  
2 2  
2





The image shows a screen sharing session with a terminal window open on a Mac desktop. The terminal window has a dark background and displays the following text:

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
2 4
2
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

Below the terminal window, the word "ASSES" is partially visible, suggesting it might be part of a larger word like "ASSESSED".

```
c a
main()
1 #include <stdio.h>
2 int main(void) {
3
4     int a[5] = {2, 4, 6, 8, 22};
5
6     int *p;
7
8     p = &(a[1]);
9
10    printf("%d %d\n", a[0], *(p--));
11    printf("%d\n", p[0]);
12
13    return 0;
14 }
15
```

\*  $a[-1]$

```
c a.c
main()
1 #include <stdio.h>
2 int main(void) {
3
4     int a[5] = {2, 4, 6, 8, 22};
5
6
7
8     printf("%d\n", a[-1]);
9
10    return 0;
11 }
12 }
```

```
Desktop -- zsh -- 63x10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c -w
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
0
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

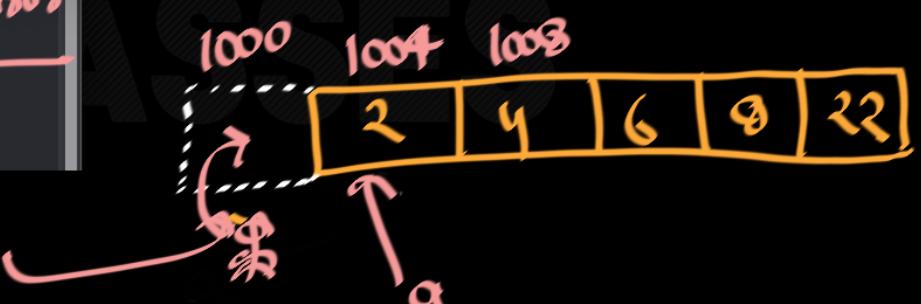
$$a[-1] \equiv 1000$$

$\ast \text{ } a[-1]$

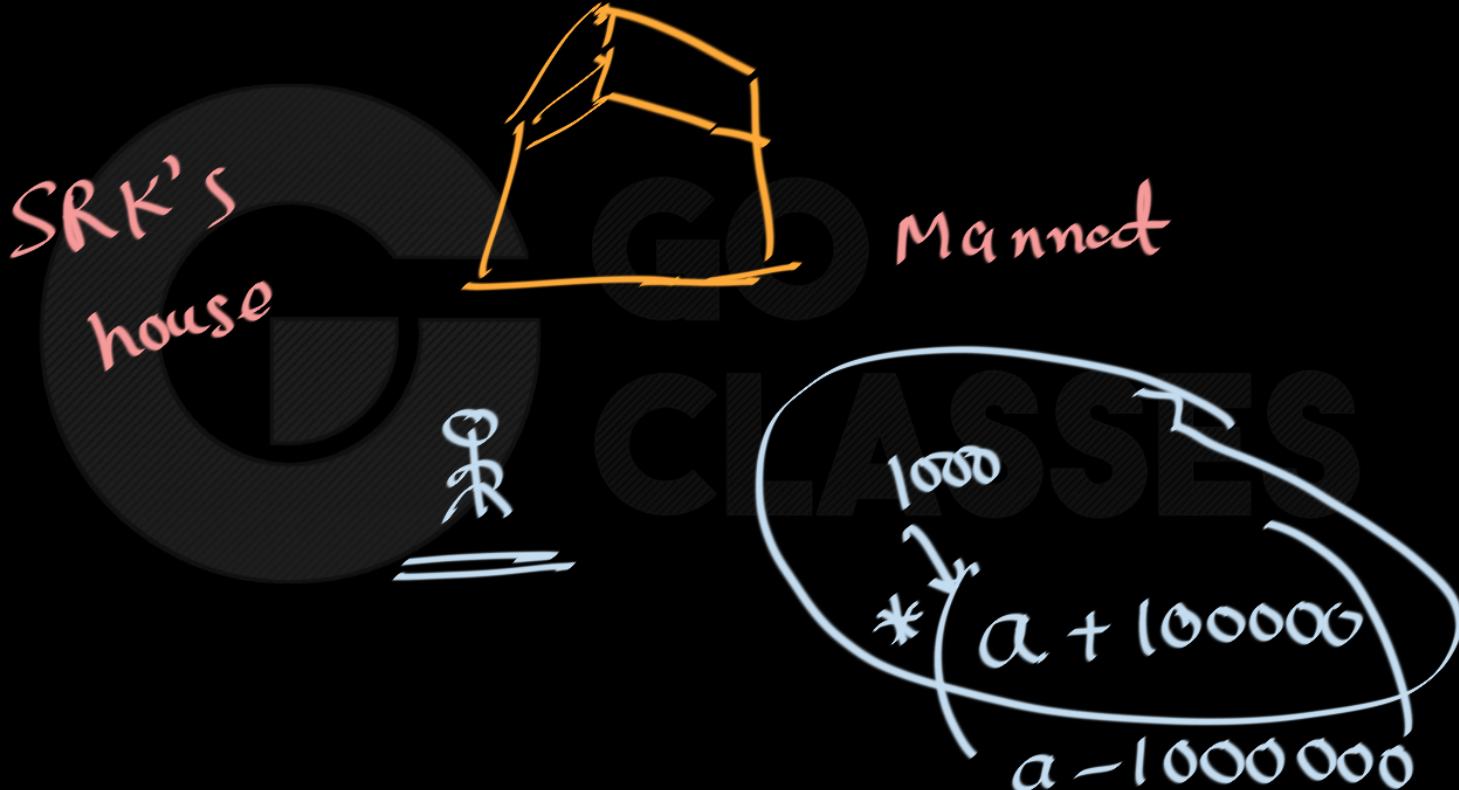


Run time error

$a[-1]$

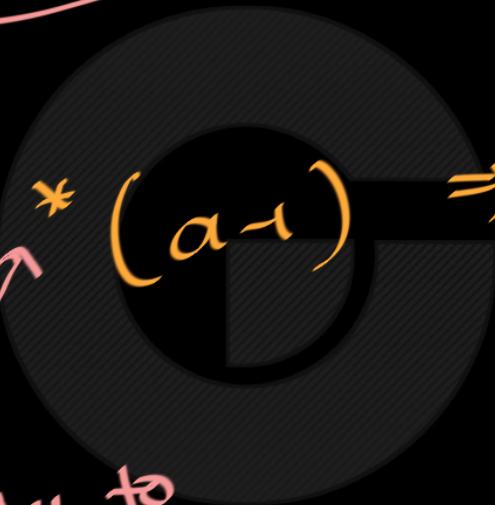


{ run time error  
(segmentation fault)  
↳ these errors occurs when we try to  
read / write into the unknown memory



address of  
SRK  
house

You try to  
enter inside  
house.



$a-1 \Rightarrow$  Not an issue

$\Rightarrow$  GO may lead to  
run time error



## Question 5:

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

```
int main() {
```

```
    int a[] = { 1, 2, 4, 8 };
```

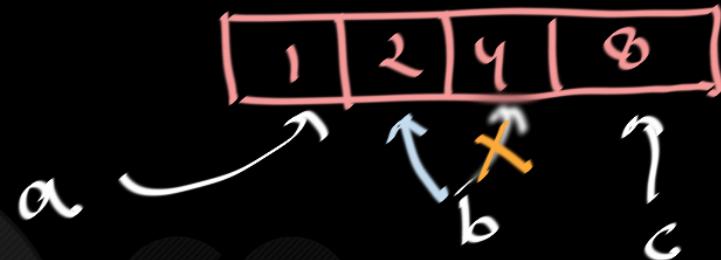
```
    int *b = a+2;
```

```
    int *c = b-- + 1;
```

```
    printf("%d ", *b);
```

```
    printf("%d", *c);
```

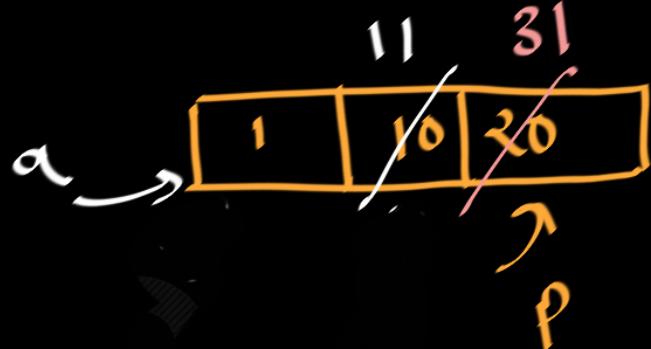
```
}
```



- A. 2 8
- B. 2 4
- C. 2 2
- D. None of these

## Question 6:

```
int a[] = {1,10,20}, *p=a, i=1, j=2;  
for (i=0, p=a; i < 2; i++) {  
    int j = *p;  p++;  *p = *p + j;  
}  
printf("out: i=%d j=%d a=%d,%d,%d\n", i, j, a[0], a[1], a[2]);
```

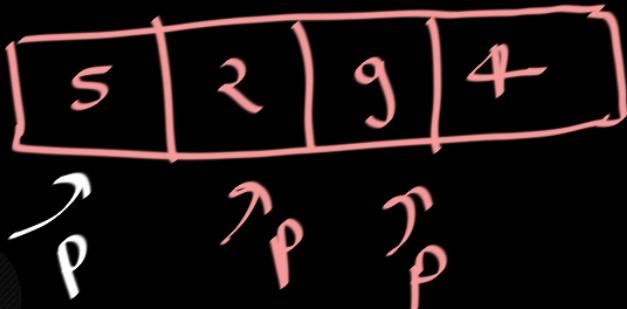


i=0 ✓  
i=1 ✓

## Question 7:

```
int main(void)
{
    int a[4] = {5, 2, 9, 4};
    int sum = 0;
    for (int *p=a; p < a+4; p++) {
        sum += *p;
    }
    printf("%d", sum);
    return 0;
}
```

~~Sum~~ ↗ ↘ ↙  
Sum



$$p = a \quad \checkmark$$

$$p = a + 1$$

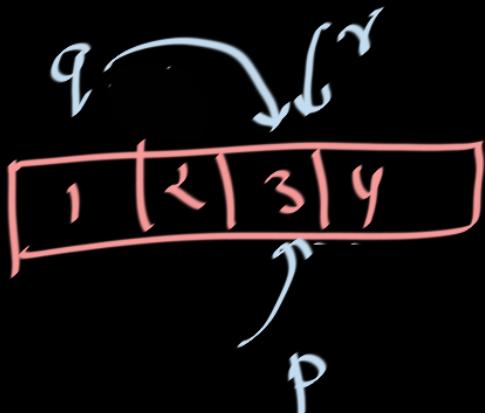
$$p = a + 2$$

$$p = a + 3$$



## Question 8:

```
int x[] = {1, 2, 3, 4};  
int *p = &x[2];  
int *q = &x[1];  
int *r = ++q;
```



```
printf("%d \n", *(p + 1));  
printf("%d \n", *(p - 1));  
printf("%d \n", *q);  
printf("%d \n", *r);
```





## Question 9:

What values does the array a contain at the end of the following code fragment?

```

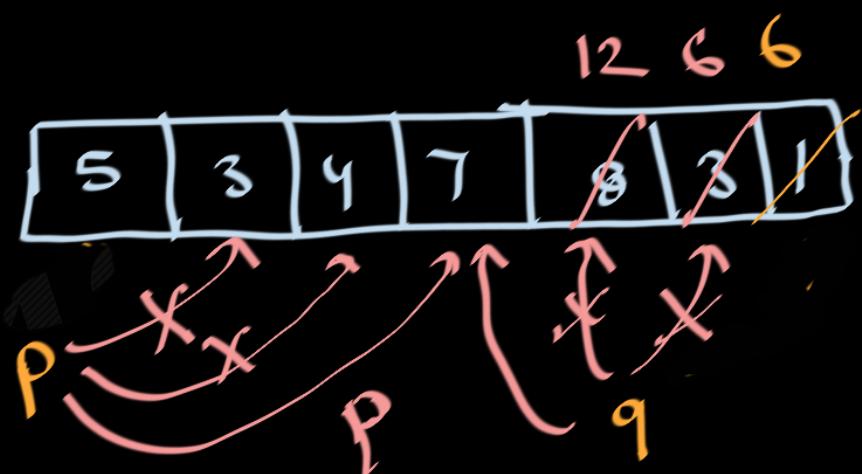
int a[7] = {5, 3, 4, 7, 8, 3, 1};

int* p = a;
int* q = &(a[6]); // address of last item

while (p != q) {
    *q += *p; → *q = *q + *p
    p++;
    q--;
}

```

a[0]: 5 a[1]: 3 a[2]: 4 a[3]: 7 a[4]: 12 a[5]: 6 a[6]: 6



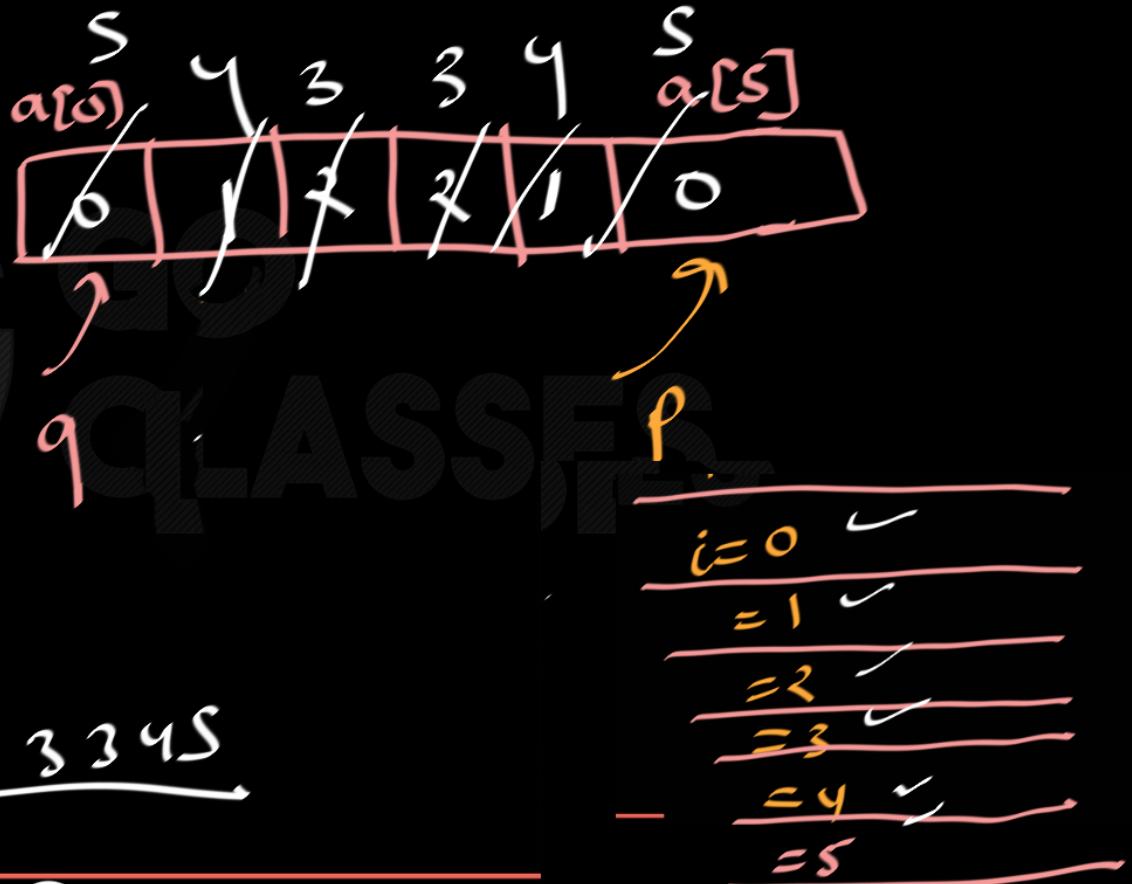
## Question 10: What does the following program print?

```
#define N 6
main() {
    int i;
    int a[N];
    int *p, *q;

    p = &a[N-1];
    q = p - (N-1);

    for (i = 0; i < N; i++) {
        *(p-i) = i;
        *(q+i) = i;
    }

    for (i = 0; i < N; i++) {
        printf("%d ", a[i]);
    }
}
```





> C a  
No Selection

```
1 #include <stdio.h>
2 #define N 6
3 main() {
4     int i;
5     int a[N];
6     int *p, *q;
7
8     p = &a[N-1];
9     q = p - (N-1);
10
11    for (i = 0; i < N; i++) {
12        *(p-i) = i;
13        *(q+i) = i;
14    }
15
16    for (i = 0; i < N; i++) {
17        printf("%d ", a[i]);
```

c a.c

Line: 21 Col: 1

Desktop -- zsh -- 63x10

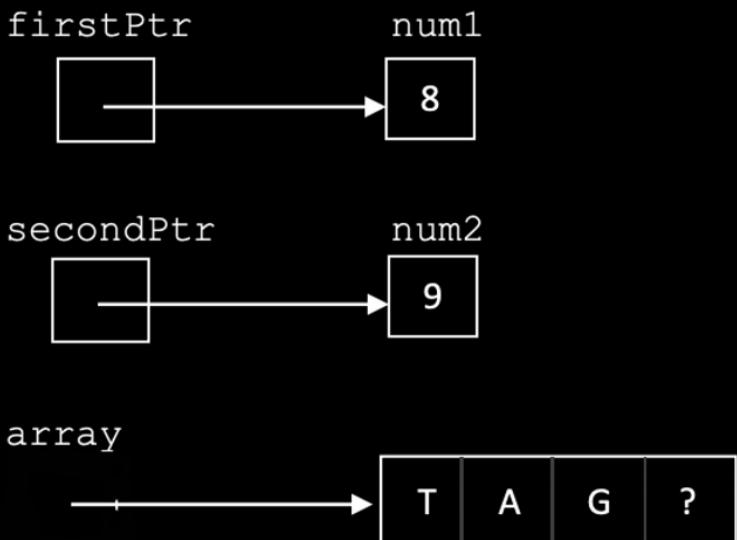
```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
5 4 3 3 4 5 %
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```



## Question 11:

The following code creates structures represented by the diagrams below:

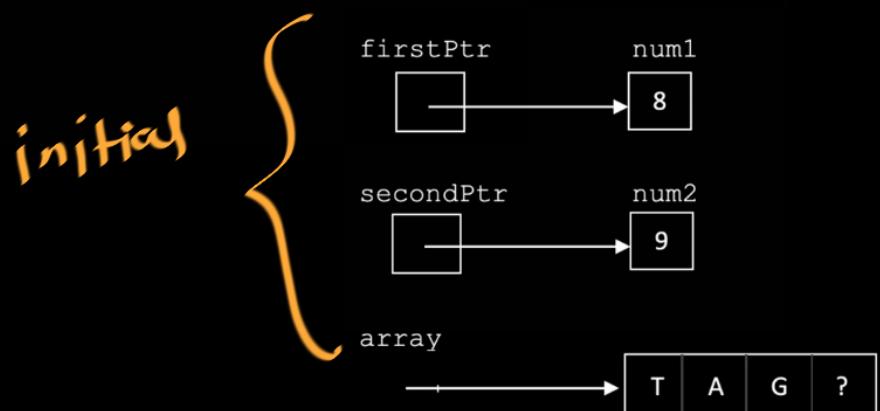
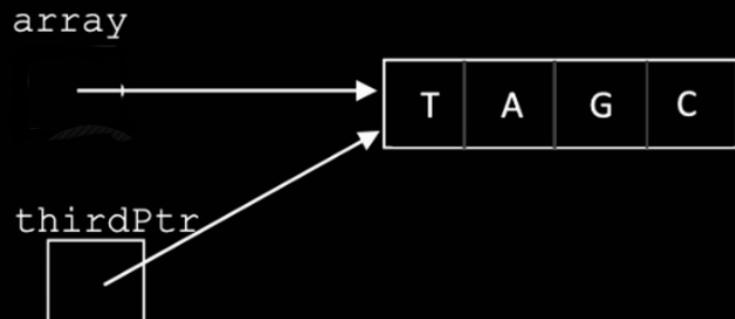
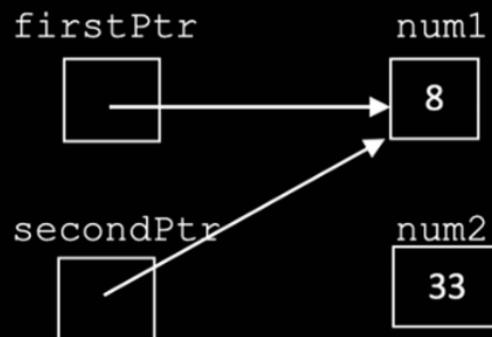
```
int num1 = 8;  
int num2 = 9;  
int *firstPtr = &num1;  
int *secondPtr = &num2;  
char array[4] = {'T', 'A', 'G', '?'};
```





# C Programming

Write the few lines of code necessary to change the above diagrams to the diagrams below, but do not use the variable names num1 and num2 in your solution.





# C Programming

Answer:

```
char * thirdPtr = array;  
*secondPtr = 33;  
secondPtr = firstPtr;  
array[3] = 'C';
```





# Pointer Arithmetic and sizeof operator



sizeof operator



## sizeof operator

( + , - , ++ , -- , / )

The **sizeof** operator is the only one in C, which is evaluated at compile time.



# C Programming

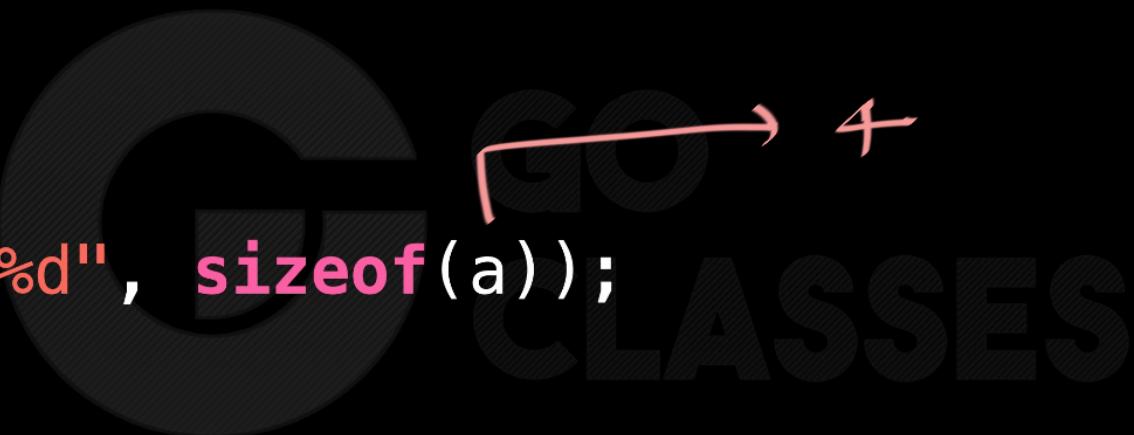
```
#include<stdio.h>
int main()
{
    printf("%d", sizeof(int));
}
```

 → 4 Bytes



# C Programming

```
#include<stdio.h>
int main()
{
    int a;
    printf("%d", sizeof(a));
}
```



`sizeof(i++)`

```
c a.c
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
4
1%
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

```
1 #include<stdio.h>
2
3 int main()
4 {
5
6
7     int i = 1;
8
9     printf("%d\n", sizeof(i++));
10    printf("%d", i);
11
12
13 }
```

Compiler Knows i++ is going to be an int



Compiler  
inside  
doesn't  
evaluate  
any  
expression  
**Sizeof** operator





The image shows a screenshot of a terminal window on a Mac OS X desktop. The terminal window has a dark background and light-colored text. At the top, it says "Desktop — zsh — 63x10". Below that, the command "(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out" is entered, followed by the output "8". The terminal window is positioned above a code editor window.

```
a.c
c a
c a > f main()
1 #include<stdio.h>
2
3 int main()
4 {
5
6
7     int a = 1;
8     long int b = 1;
9
10    printf("%d\n", sizeof(b));
11
12
13
14 }
```



# sizeof(Expression)

Example :-

**sizeof(long integer + integer)**, **sizeof(char + integer)**, **sizeof(char + char)**



A screenshot of a Mac desktop environment. On the left, a code editor window titled 'a.c' shows a C program with line numbers from 1 to 14. The code includes standard input/output headers, a main function declaration, variable declarations for 'a' and 'b', and a printf statement to output the size of their sum. On the right, a terminal window titled 'Desktop -- zsh - 63x10' shows the command 'gcc a.c -w' being run, followed by the execution of the compiled program './a.out', which outputs the value '8'.

```
#include<stdio.h>
int main()
{
    int a = 1;
    long int b = 1;
    printf("%d\n", sizeof(a+b));
```

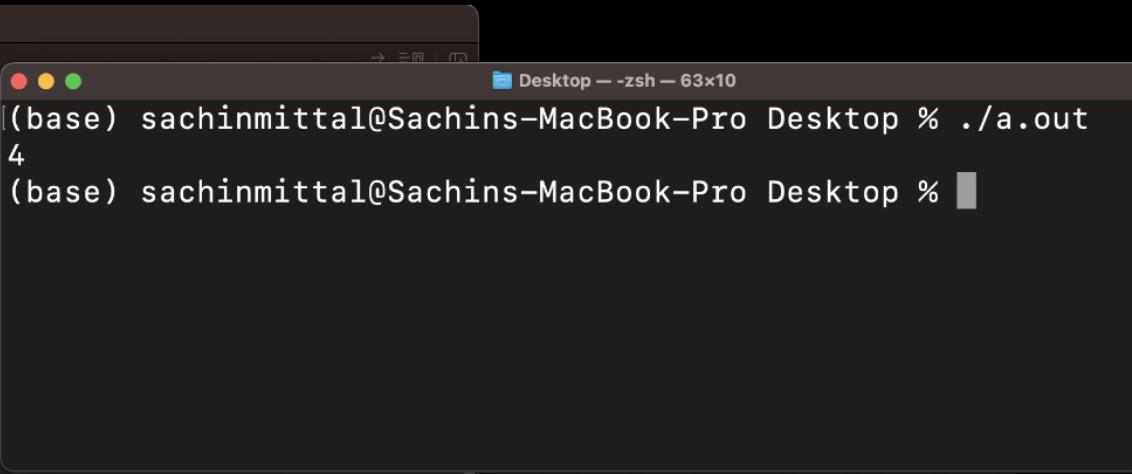


The image shows a screenshot of a terminal window titled 'Desktop — zsh — 63x10'. The window displays the following text:

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c -w
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
4
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

To the left of the terminal window, there is a code editor window titled 'a.c'. The code editor contains the following C program:

```
1 #include<stdio.h>
2
3 int main()
4 {
5
6     char c = 'a';
7     int a = 1;
8     long int b = 1;
9
10    printf("%d\n", sizeof(c+a));
11
12
13
14 }
```

optional ( integer Promotion )

```
a.c
c a
main()
1 #include<stdio.h>
2
3 int main()
4 {
5
6     char c = 'a';
7     char d = 'b';
8
9     int a = 1;
10    long int b = 1;
11
12    printf("%d\n", sizeof(c+d));
13
14
15
16 }
```

(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out  
4  
(base) sachinmittal@Sachins-MacBook-Pro Desktop %



# C Programming

## sizeof(array\_name)

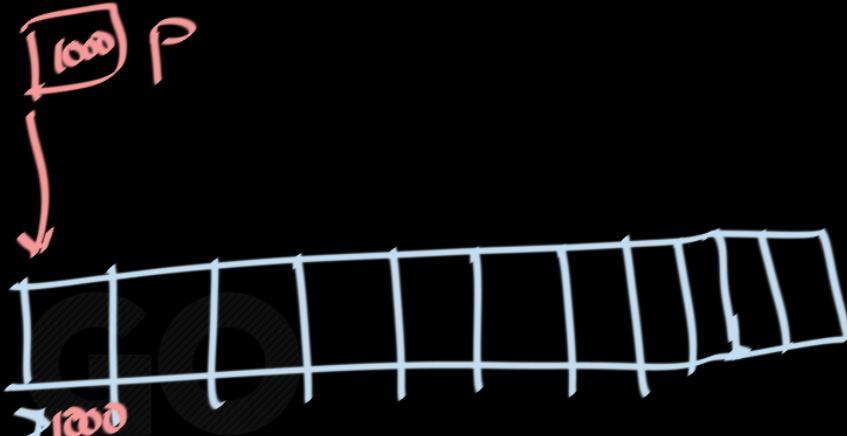
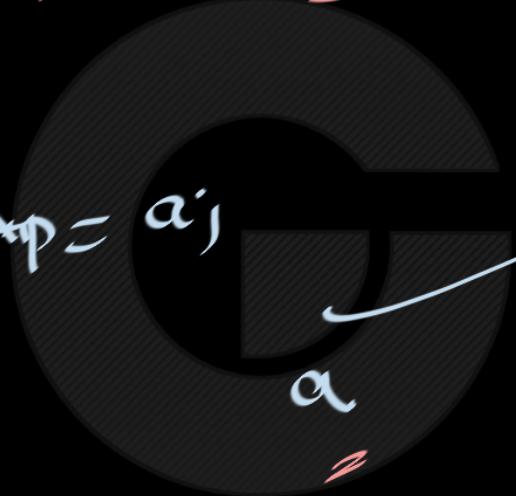
```
a.c
main()
1 #include<stdio.h>
2
3 int main()
4 {
5     int a[10];
6
7     printf("%d\n", sizeof(a));
8
9
10
11
12 }
13
```

Desktop --zsh - 63x10  
[(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c -w  
[(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out  
40  
[(base) sachinmittal@Sachins-MacBook-Pro Desktop % ]

No. of elements x size of one element

```
int a [10];
```

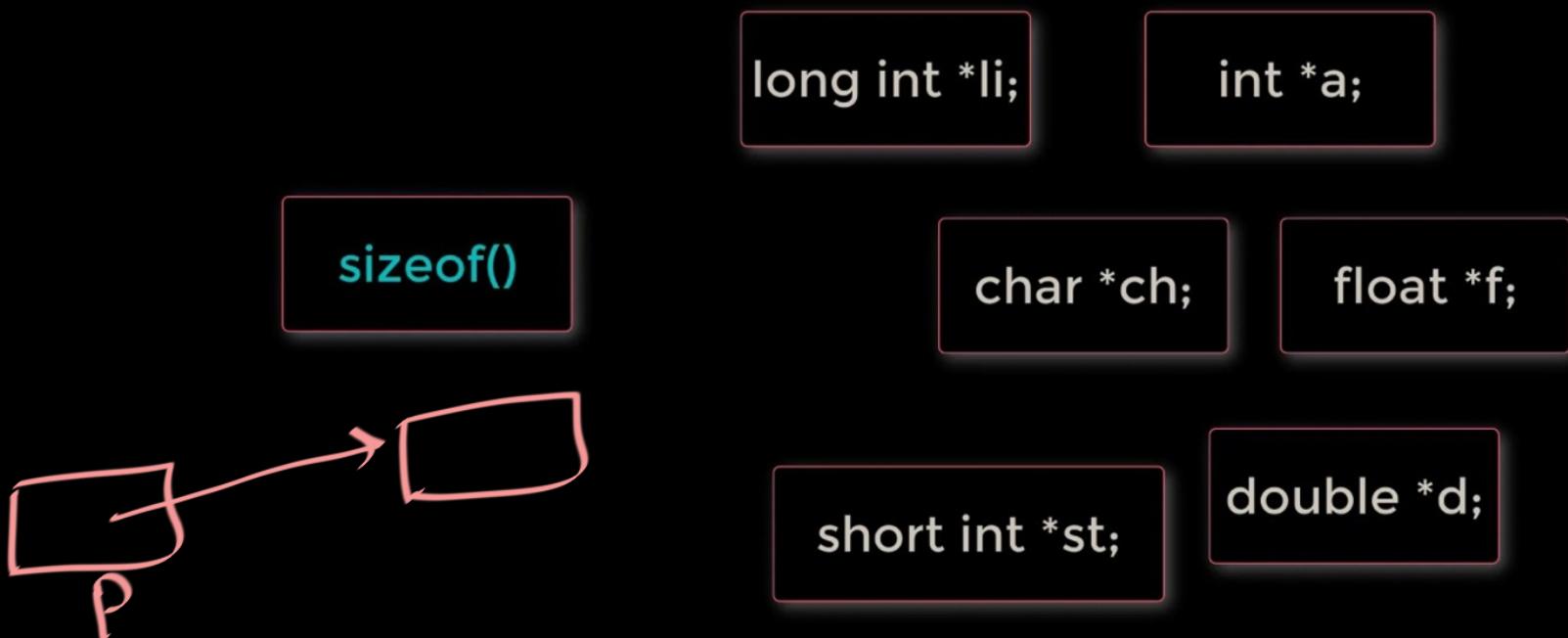
```
int *p = a;
```



# CLASSES



**sizeof(address) or sizeof(pointer)**



A screenshot of a Mac desktop environment. On the left, a code editor window titled 'a.c' shows a C program with line numbers from 1 to 14. The program prints the size of a character pointer. On the right, a terminal window titled 'Desktop - zsh - 63x10' shows the output of running the program: the number 8. The background of the desktop is a dark gray color with large, semi-transparent white text that reads 'CLASSES' vertically.

```
1 #include<stdio.h>
2
3 int main()
4 {
5
6     char *t;
7     printf("%d\n", sizeof(t));
8
9
10 }
11
12
13
14
```

```
[base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
8
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```



The image shows a screenshot of a terminal window on a Mac OS X desktop. The terminal title is 'Desktop -- zsh -- 63x10'. The command entered was 'gcc a.c -w' followed by './a.out'. The output of the program is '8', which is the size of a character variable in bytes. The terminal window has a dark background with light-colored text. The desktop background is visible behind the terminal window, showing a large watermark of the word 'CLASSES'.

```
a.c
main()
1 #include<stdio.h>
2
3 int main()
4 {
5
6     char *t;
7     char c;
8
9     printf("%d\n", sizeof(&c));
10
11
12
13 }
```

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c -w
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
8
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```



A screenshot of a Mac OS X desktop environment. On the left, there's a code editor window titled 'a.c' containing a simple C program. On the right, there's a terminal window titled 'Desktop -- zsh -- 63x10' showing the execution of the program and its output.

```
No Selection
1 #include<stdio.h>
2
3 int main()
4 {
5
6     int i;
7
8     printf("%d\n", sizeof(&i));
9
10
11 }
12
13
14
15
```

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
8
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```



The image shows a split-screen environment. On the left is a code editor window titled 'c a.c' containing a C program. On the right is a terminal window titled 'Desktop - zsh - 63x10' showing the execution and output of the program.

**Code Editor (Left):**

```
1 #include<stdio.h>
2
3 int main()
4 {
5
6     char *t;
7     char c;
8
9     int *p;
10    int i;
11
12    printf("%d\n", sizeof(p));
13
14
15 }
```

**Terminal (Right):**

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c -w
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
8
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

```
a.c
No Selection
1 #include<stdio.h>
2
3 int main()
4 {
5
6     char *t;
7     printf("%d\n", sizeof(t));
8
9
10 }
11
12
13
```

(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out  
8  
(base) sachinmittal@Sachins-MacBook-Pro Desktop %

```
c.a
main()
1 #include<stdio.h>
2
3 int main()
4 {
5
6     char *t;
7     char c;
8
9     int *p;
10    int i;
11
12    printf("%d\n", sizeof(p));
13
14
15 }
16
```

(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c -w  
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out  
8  
(base) sachinmittal@Sachins-MacBook-Pro Desktop %

GO  
SSES



# C Programming

```
int a[5];  
printf("%d, %d, %d \n", sizeof(int), sizeof(a[3]), sizeof(a));
```

$$5 \times 4 = 20$$



# C Programming

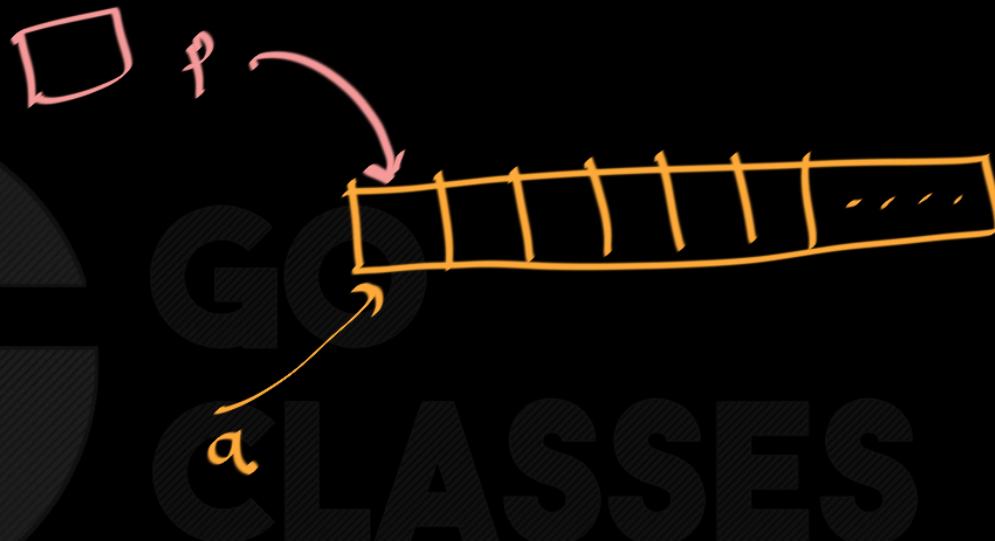
```
int *p; //pointer (address) where a value of int type is stored  
int a[10]; //a continuous block of memory for 10 int values  
sizeof(p); //no.of bytes for storing the address (8 for 64-bit)  
sizeof(a); //size of the allocated array is 10*sizeof(int)
```

①

②

```
int a[10];
int *p;

p = a;
```





A screenshot of a terminal window titled 'a.c'. The window shows the command '(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out' followed by the output '8' and '40'. The background of the terminal window is dark, and the text is white.

```
1 #include<stdio.h>
2
3 int main()
4 {
5
6     int a[10];
7     int *p;
8
9     p = a;
10
11
12     printf("%d\n", sizeof(p));
13     printf("%d\n", sizeof(a));
14
15
16
17 }
```



Pointer  
to  
address

GO  
CLASSES



# GO CLASSES

## Pointer Arithmetic



# Pointer Arithmetic

**(*a+i*)** is the pointer arithmetic

**i=1** results in the following “assembler code”:

***a + sizeof(one element)***

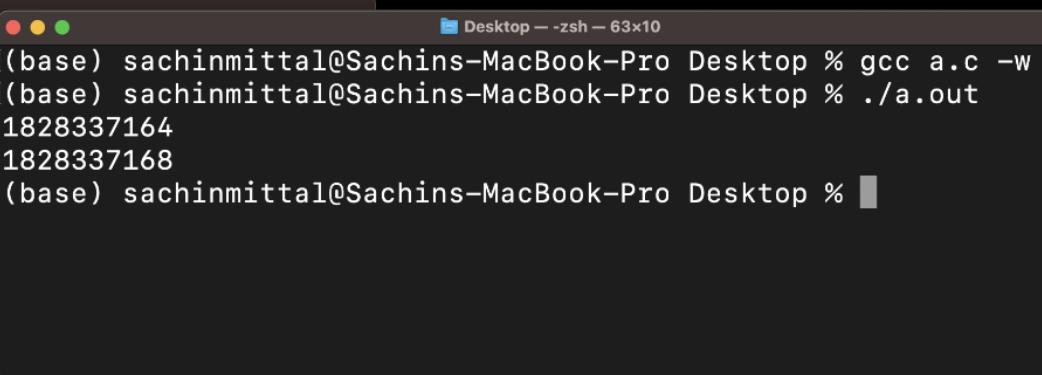
*a+1*  
SES

Otherwise the “assembler code” is:

***a + i \* sizeof(one element)***

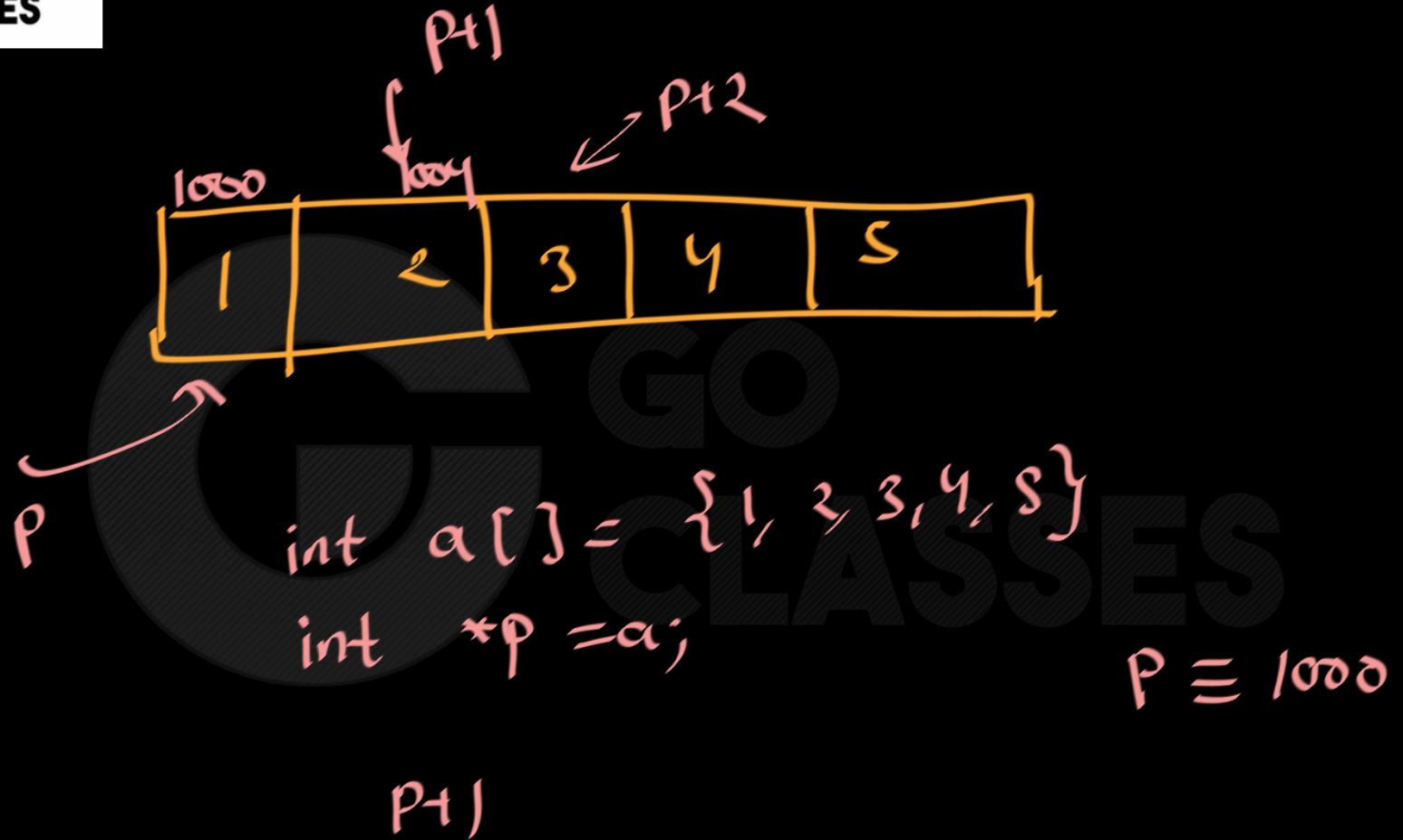
int x;

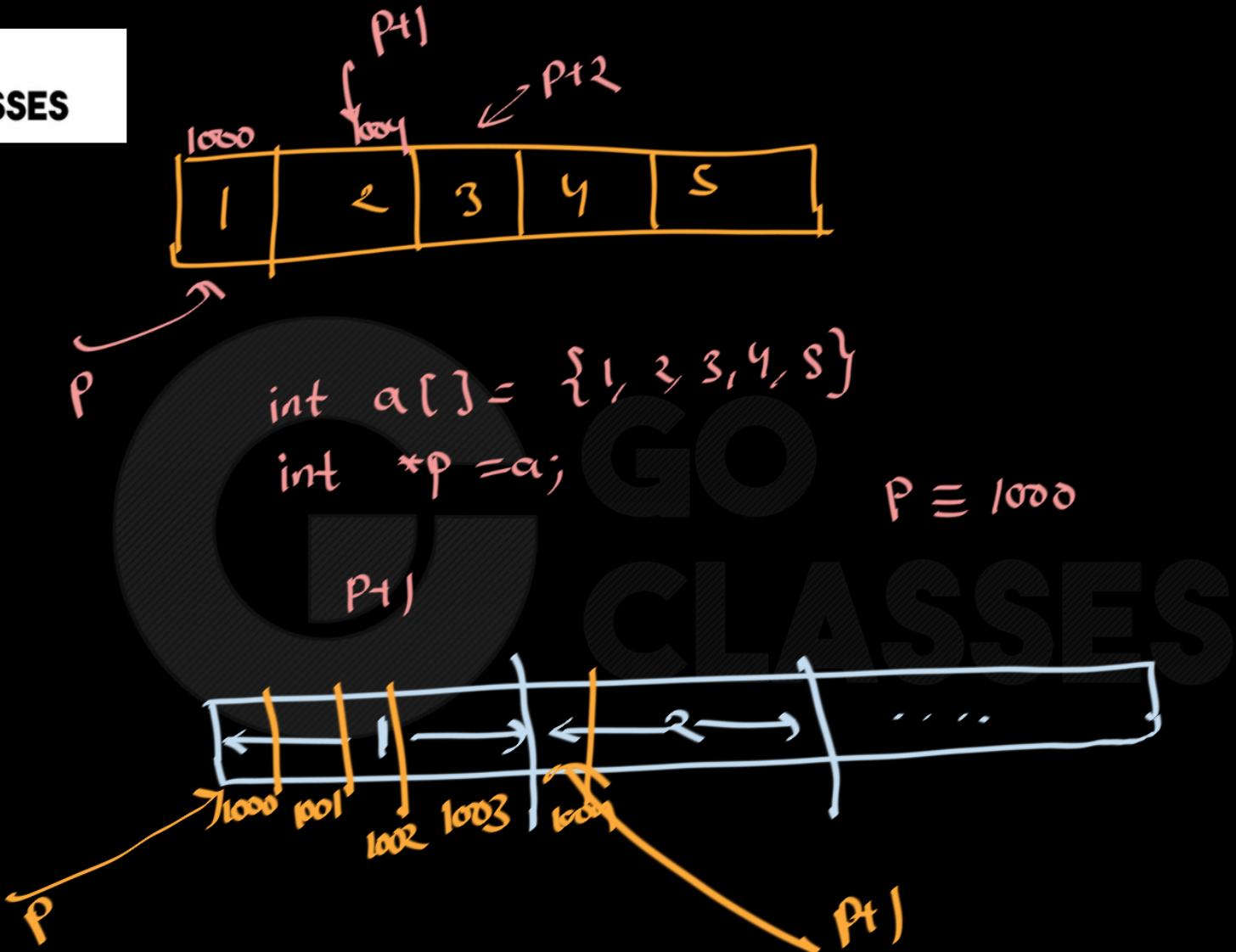
int \*p = &x;



```
a.c
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c -w
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
1828337164
1828337168
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int x;
6     int *p = &x;
7
8     printf("%u\n", p);
9     printf("%u\n", p+1);
10
11
12 }
```





A screenshot of a Mac desktop environment. On the left, a code editor window titled 'a.c' shows a C program. The code defines an array 'a' and a pointer 'p' to its first element. It then prints the values at memory locations 'p' and 'p+1' using printf. On the right, a terminal window titled 'Desktop -- zsh -- 63x10' shows the output of running the compiled program, which is the same as the input shown in the code editor.

```
#include <stdio.h>
int main()
{
    int a[] = {1,2,3,4,5};
    int *p = a;

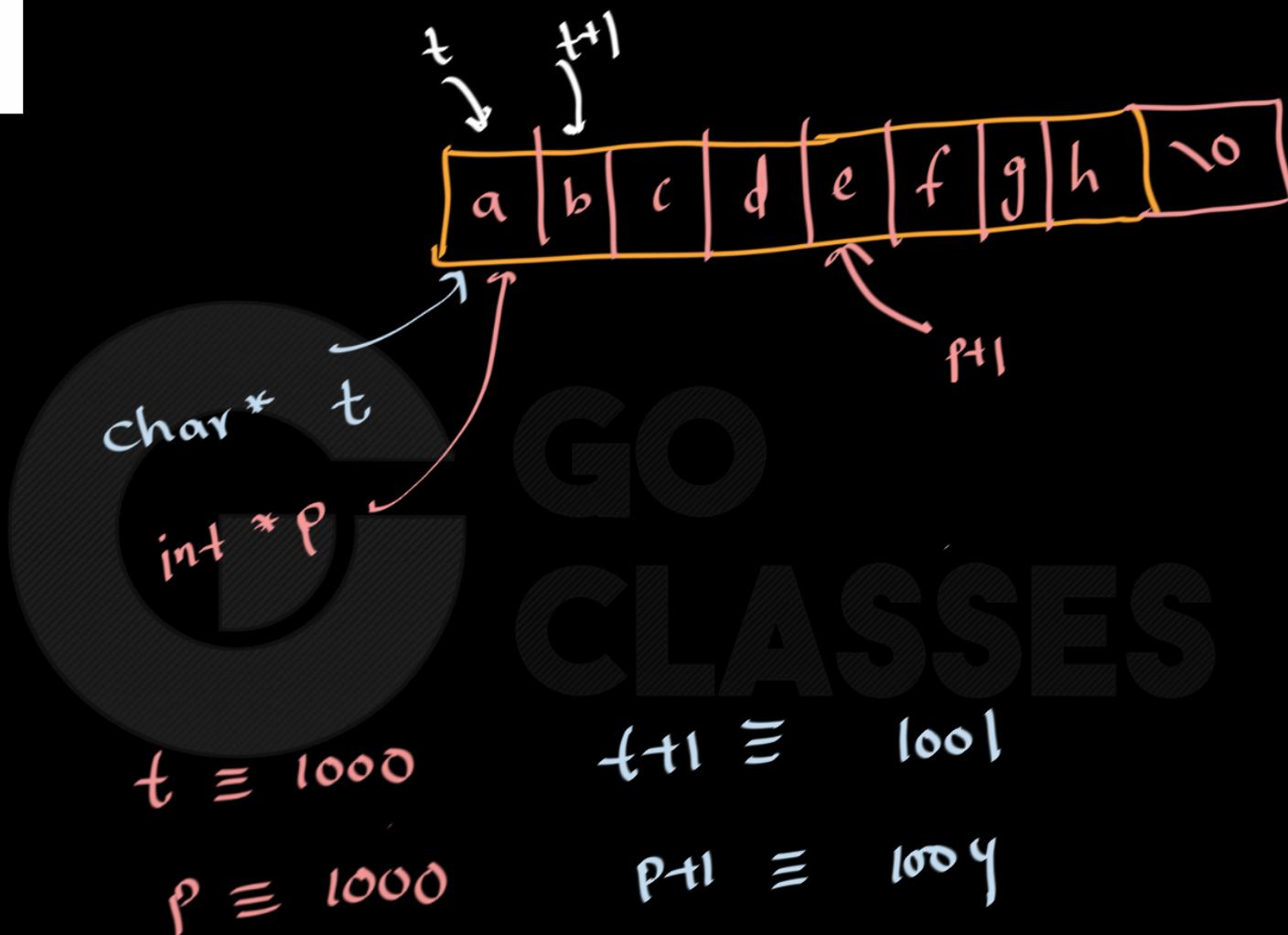
    printf("p = %u\n", p);
    printf("p+1 = %u\n", p+1);

    printf("*p = %u\n", *p);
    printf("*(p+1) = %u\n", *(p+1));
}
```

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c -w
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
p = 1796453872
p+1 = 1796453876
*p = 1
*(p+1) = 2
```

```
c a
main()
1 #include <stdio.h>
2
3 int main()
4 {
5     char s[] = "abcdefgh";
6     char *t = s;
7
8     printf("t = %u\n", t);
9     printf("t+1 = %u\n", t+1);
10
11
12    printf("*t = %c\n", *t);
13    printf("*(t+1) = %c\n", *(t+1));
14
15 }
```

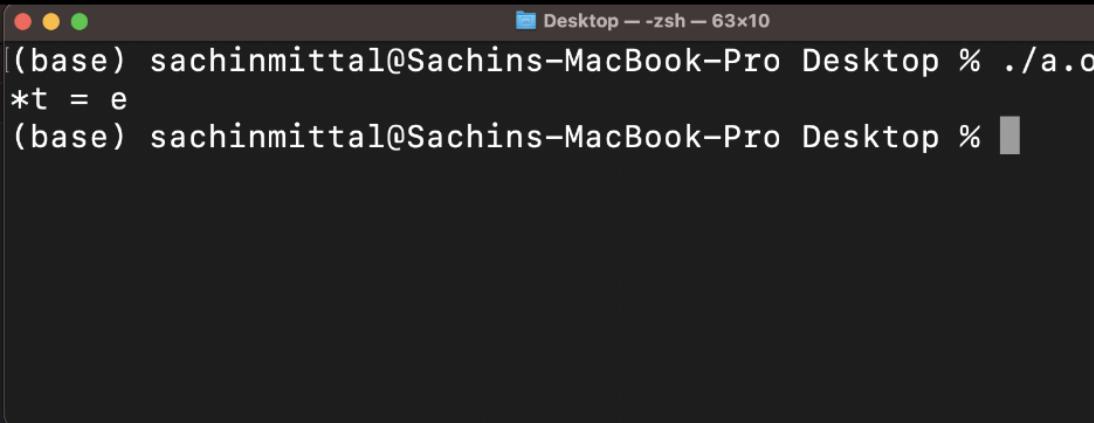






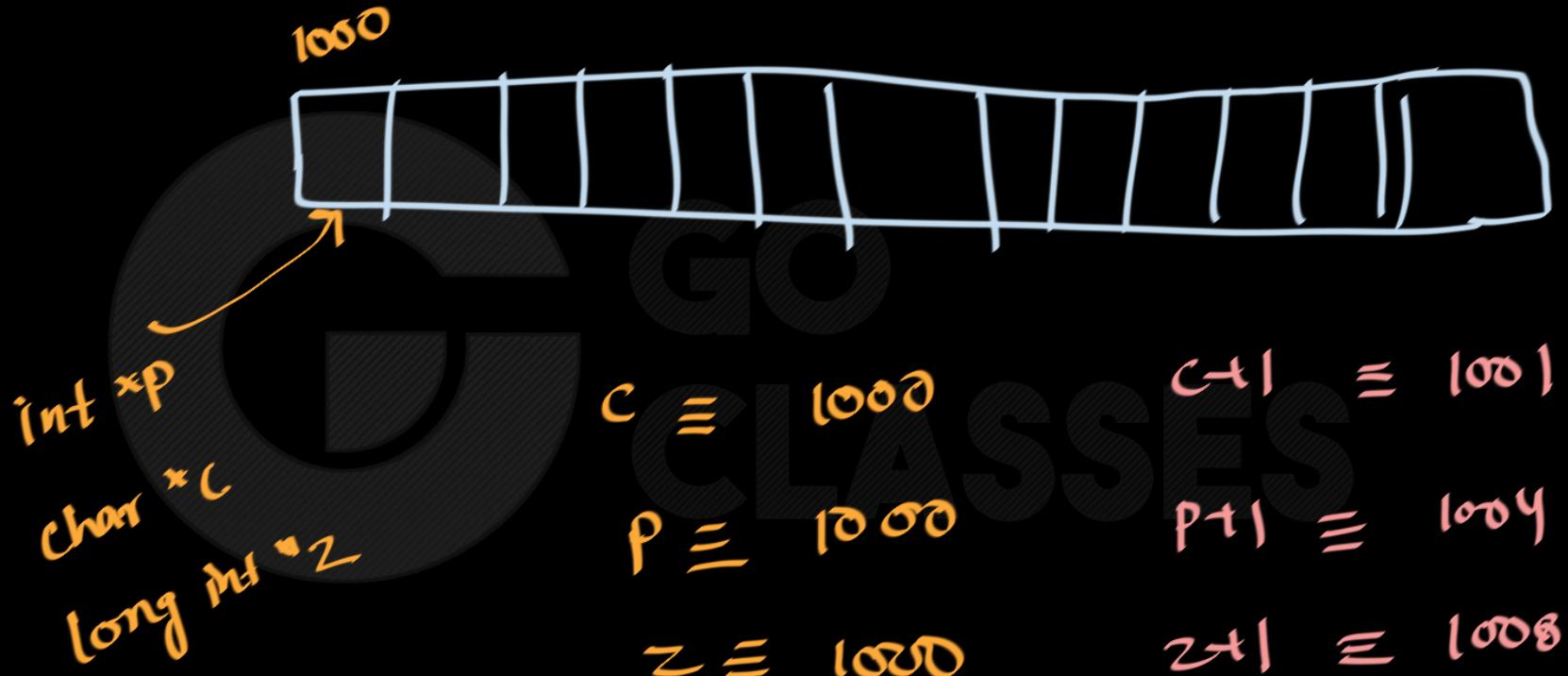
```
c a.c
a > No Selection
1 #include <stdio.h>
2
3 int main()
4 {
5     char s[] = "abcdefg";
6     char *t = s;
7     int *p = s; //not recommended
8
9     printf("t = %u\n", t);
10    printf("t+1 = %u\n", t+1);
11
12    printf("p = %u\n", p);
13    printf("p+1 = %u\n", p+1);
14 }
```

```
Desktop -- zsh -- 63x10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a
t = 1860908536
t+1 = 1860908537
p = 1860908536
p+1 = 1860908540
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```



The image shows a terminal window on a Mac OS X desktop. The title bar says "Desktop -- zsh -- 63x10". The command entered is ". ./a.o". The output is: "(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.o \*t = e (base) sachinmittal@Sachins-MacBook-Pro Desktop %".

```
2
3 int main()
4 {
5     char s[] = "abcdefgh";
6     char *t = s;
7     int *p = s; //not recommended
8
9     // t = (char *) (p+1);
10    t = p+1;
11    printf("*t = %c\n", *t);
12
13 }
```



$\text{int } *P$

$$P+1 \equiv P + \text{sizeof(int)}$$

$\text{char } *t$

$$t+1 \equiv t + \text{sizeof(char)}$$

$\text{long int } *z$

$$z+1 \equiv z + \text{sizeof(long int)}$$

optional

$$q+1 \equiv q + \text{sizeof}(*q) \rightarrow \begin{matrix} \text{type that} \\ q \text{ points to} \end{matrix}$$



# Valid arithmetic on pointers (*addresses*)

1. Adding or Subtracting an integer to the pointer

2. **Difference between two pointers**  
**(Tells the distance in terms of number of elements)**

3. Comparing pointers

$$\begin{array}{l} P_1 - P_2 \\ P_1 > P_2 \quad P_1 \neq P_2 \end{array}$$

**NOTE: Generally we do pointer arithmetic only on elements of arrays**



# C Programming



```
2
3 int main()
4 {
5
6     int a[] = {1,2,3,4,5};
7     int *p = a;
8     int *q = a+3;
9
10    printf("*p = %u\n", *p);
11    printf("*q = %u\n", *q);
12
13
14
15 }
```

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.o  
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.o  
*p = 1  
*q = 4  
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```



```
2
3 int main()
4 {
5     int a[] = {1,2,3,4,5};
6     int *p = a;
7     int *q = a+3;
8
9     printf("*q = %u\n", *(q-1));
10
11
12
13
14 }
```

A screenshot of a Mac desktop environment. In the foreground, there is a terminal window titled 'Desktop — zsh — 63x10'. The window shows the command 'gcc a.c' being run, followed by the output of the program which prints the value '3'. The background shows the Mac OS X desktop with a dark theme, and a dock with various application icons at the bottom.



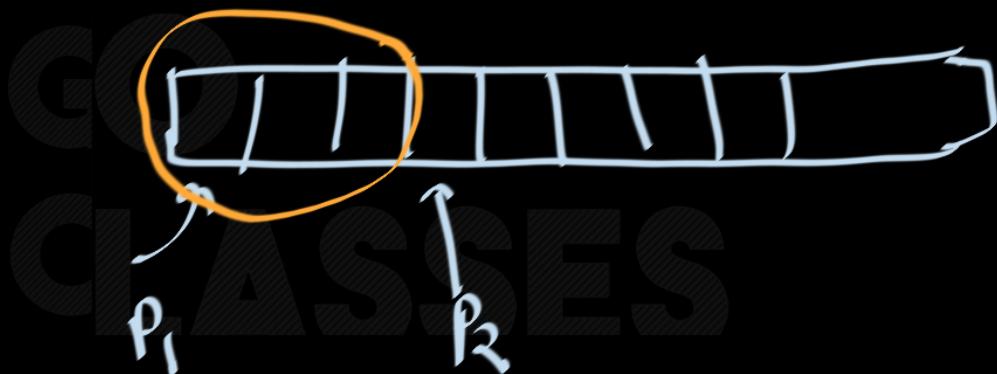
## Difference between two pointers -

```
int a[10];
int *p1 = a;
int *p2 = p1+3;
```

$$p_2 = p_1 + 3$$

```
printf("%d", p2-p1);
```

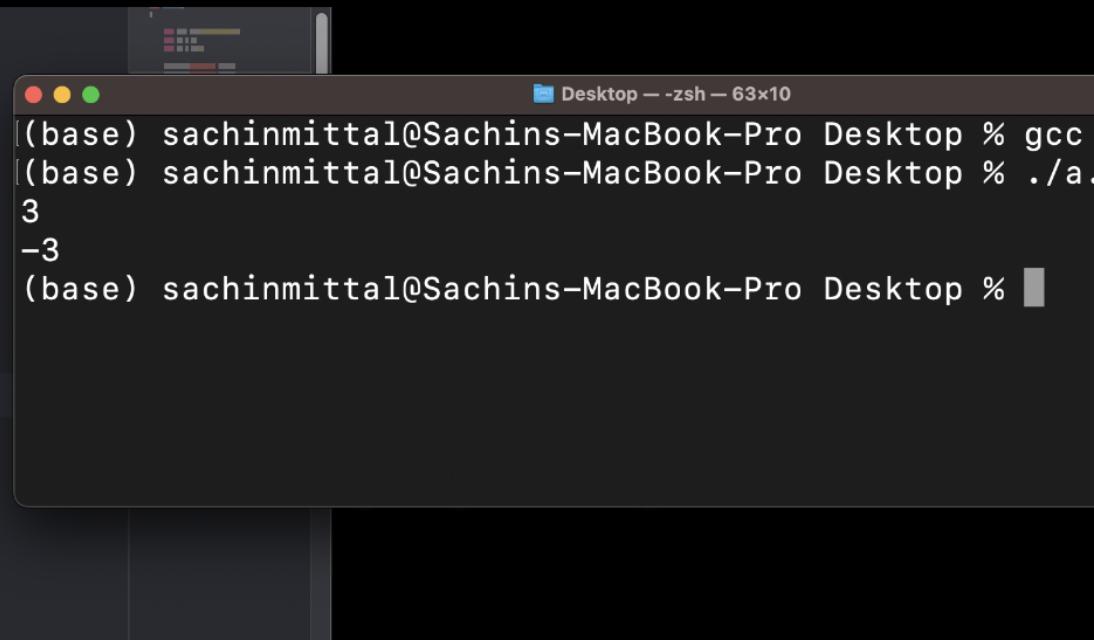
$$p_1 - p_2 = -3$$



When two pointers are subtracted, both ***shall point to elements of the same array object.***

*If not then result can be any garbage value or error.*

```
3 int main()
4 {
5     int a[] = {1,2,3,4,5};
6     int *p = a;
7     int *q = a+3;
8
9     printf("%d\n", q-p);
10    printf("%d\n", p-q);
11 }
```



```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a
3
-3
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

$$P_2 - P_1 = \frac{P_2 - P_1}{\text{Sizeof(int)}} = \frac{1012 - 1000}{4} \\ = \frac{12}{4} = 3$$

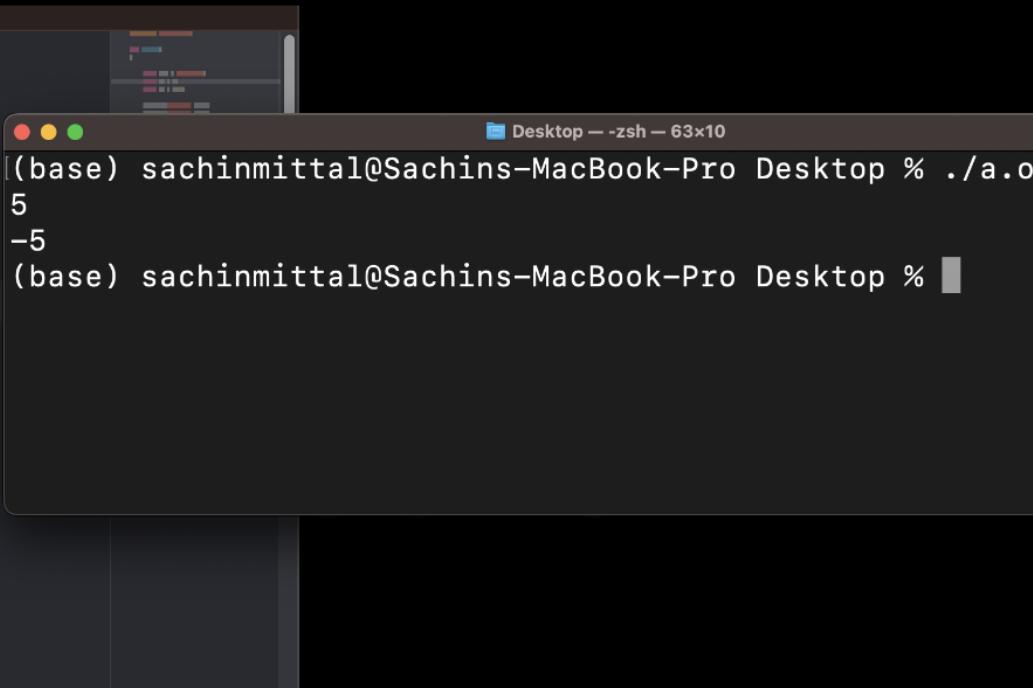
$$\text{int } *k = P_1 + 3$$

$$P_1 \equiv 1000$$

$$P_1 + 3 \equiv 1000 + 3 \times 4 = 1012$$



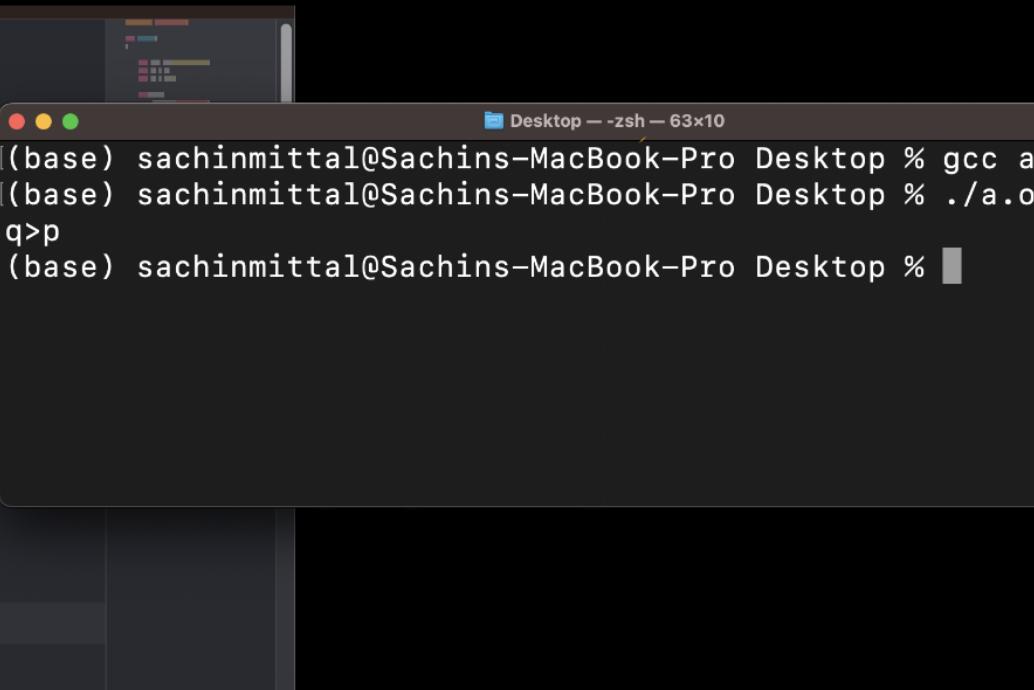
```
> main()
1 #include <stdio.h>
2
3 int main()
4 {
5
6     char s[] = "abcdef";
7     char *p = s;
8     char *q = s+5;
9
10    printf("%d\n", q-p);
11    printf("%d\n", p-q);
12
13
14
15
16 }
```



A screenshot of a terminal window titled 'Desktop -- zsh -- 63x10'. The window shows the execution of a C program named 'a.o'. The output consists of two lines of text: '5' and '-5'. The terminal has a dark background with light-colored text and standard OS X window controls (red, yellow, green) at the top.

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.o
5
-5
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int a[] = {1,2,3,4,5};
6     int *p = a;
7     int *q = a+3;
8
9     if(p>q){
10         printf("p>q \n");
11     }
12
13     if(q>p){
14         printf("q>p \n");
15     }
16 }
```



The terminal window shows the following session:

```
Desktop -- zsh -- 63x10
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.o
q>p
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```



```
#include <stdio.h>

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

    for(int *p = a; p<a+5; p++){
        printf("%d \n",*p);
    }

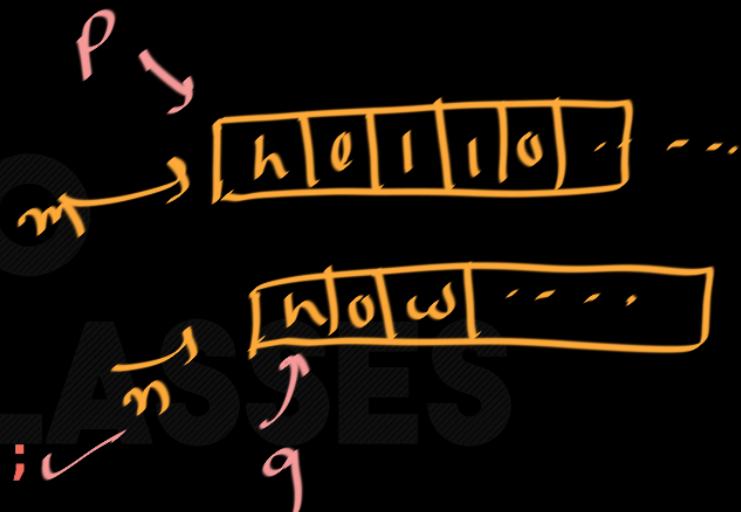
}
```

A screenshot of a macOS terminal window titled "Desktop - zsh - 72x14". The window shows the execution of a C program. The command "gcc a.c" is run, followed by "./a.out". The output displays the integers 1, 2, 3, 4, and 5, each on a new line. The terminal window has a dark theme with light-colored text and icons.



# C Programming

```
#include <stdio.h>
int main()
{
    char m[10] = "hello";
    char n[] = "how are you?";
    char *p, *q;
    p=&m[0];
    q=n;
    if ( *p == *q )
        printf("values are same\n");
    if ( p == q )
        printf("addresses are same\n");
    else
        printf("addresses differ\n");
}
```



Output  
values are same  
addresses differ



# Invalid arithmetic on pointers

p1 + p2

adding 2 addresses doesn't

p1 \* p2

make sense

p1 % p2

p1 / p2



# Pointer Arithmetic Question

How many of the following are invalid?

- I. pointer + integer (ptr+1)
- II. integer + pointer (1+ptr)
- III. pointer + pointer (ptr + ptr)
- IV. pointer – integer (ptr – 1)
- V. integer – pointer (1 – ptr)
- VI. pointer – pointer (ptr – ptr)
- VII. compare pointer to pointer (ptr == ptr)
- VIII. compare pointer to integer (1 == ptr)
- IX. compare pointer to 0 (ptr == 0)
- X. compare pointer to NULL (ptr == NULL)

**#invalid**

- A: 1
- B: 2
- C: 3
- D: 4
- E: 5

ES



## Answer:

How many of the following are invalid?

- I. pointer + integer (ptr+1)
- II. integer + pointer (1+ptr)
- III. pointer + pointer (ptr + ptr)
- IV. pointer – integer (ptr – 1)
- V. integer – pointer (1 – ptr)
- VI. pointer – pointer (ptr – ptr)
- VII. compare pointer to pointer (ptr == ptr)
- VIII. compare pointer to integer (1 == ptr)
- IX. compare pointer to 0 (ptr == 0)
- X. compare pointer to NULL (ptr == NULL)

#invalid

- A: 1
- B: 2
- C: 3
- D: 4
- E: 5



## Question

```
#include <stdio.h>
int main(void) {
    int a[8] = {2,3,4,5,6,7,8,9};
    int *p, *q, i;

    p = &(a[2]);
    q = p + 3;
    p += 4;
    q = q - 2;
    i = q - p;
    i = p - q;
    if (p<=q) printf("less\n");
    else printf("more\n"); //printed
    return 0;
}
```

ES



## Question

```
#include <stdio.h>
int main(void) {
    int a[8] = {2,3,4,5,6,7,8,9};
    int *p, *q, i;

    p = &(a[2]); // p points to a[2]
    q = p + 3; // q points to a[5]
    p += 4; // p points to a[6]
    q = q - 2; // q points to a[3]
    i = q - p; // i = 3 - 6 = -3
    i = p - q; // i = 6 - 3 = 3
    if (p<=q) printf("less\n");
    else printf("more\n"); //printed
    return 0;
}
```