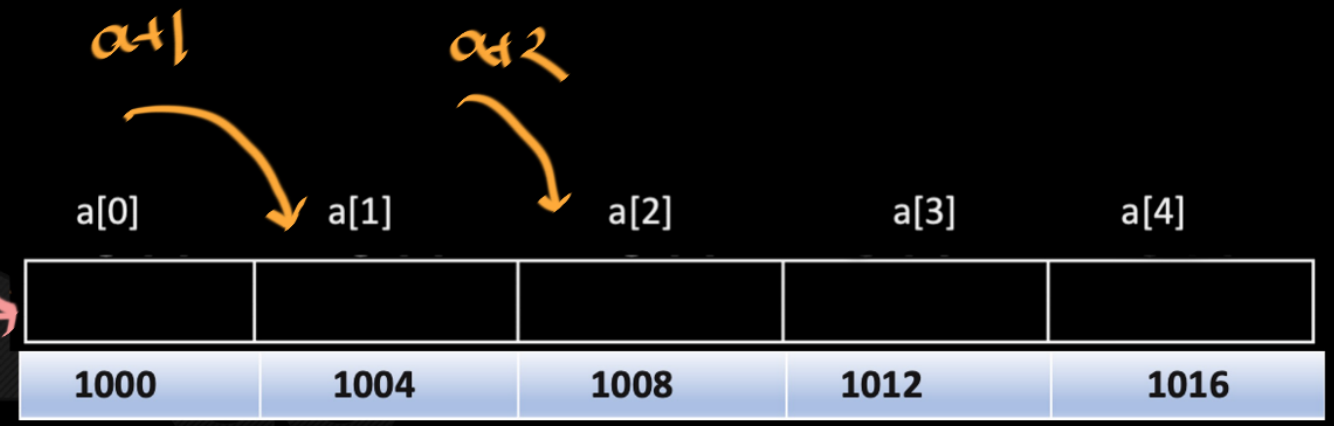
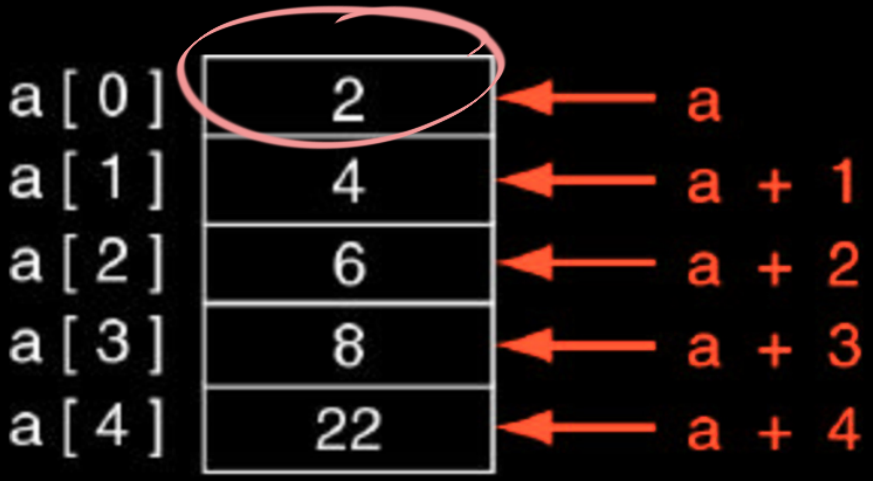
Arrays and Pointers

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



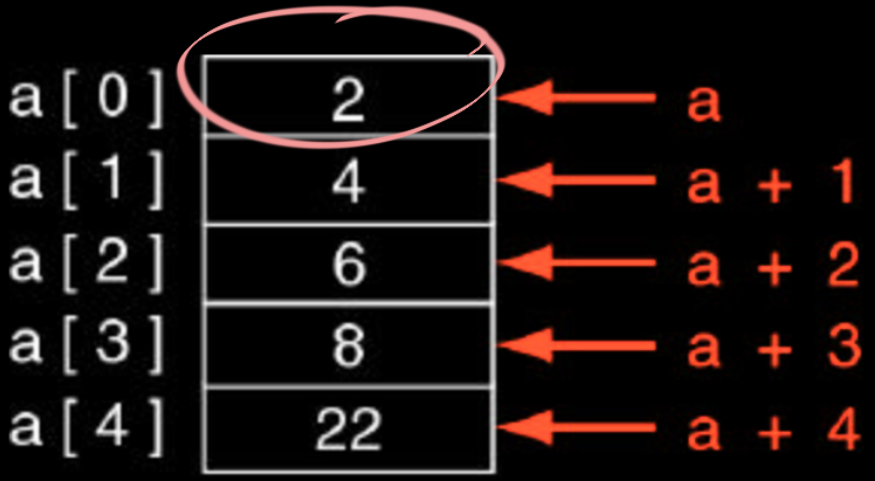


$a \equiv \&a[0]$

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

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

$a[0]$ $\quad$ $*(a+0)$



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

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

$a[0] \quad *(a+0)$

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)

Question:

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

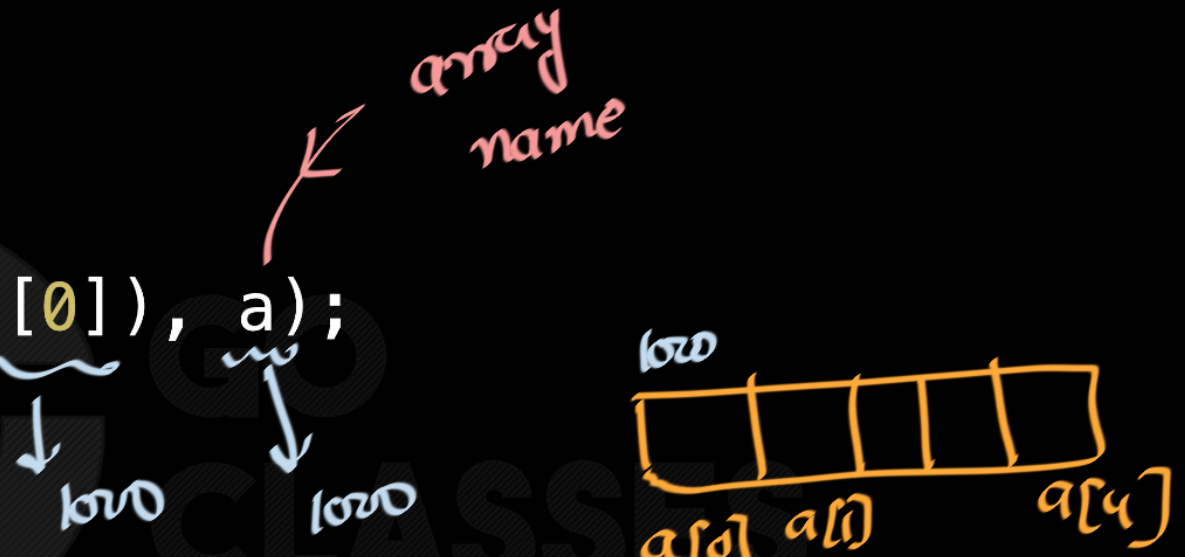
A. *a+i
B. a+i
C. *(a+i)
D. none of the above

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

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



The two values printed are identical

GO CLASSES

Small detour

%p & %u

%p
↓
designated formed specifier to print addresses.

%u → this is for unsigned integers.

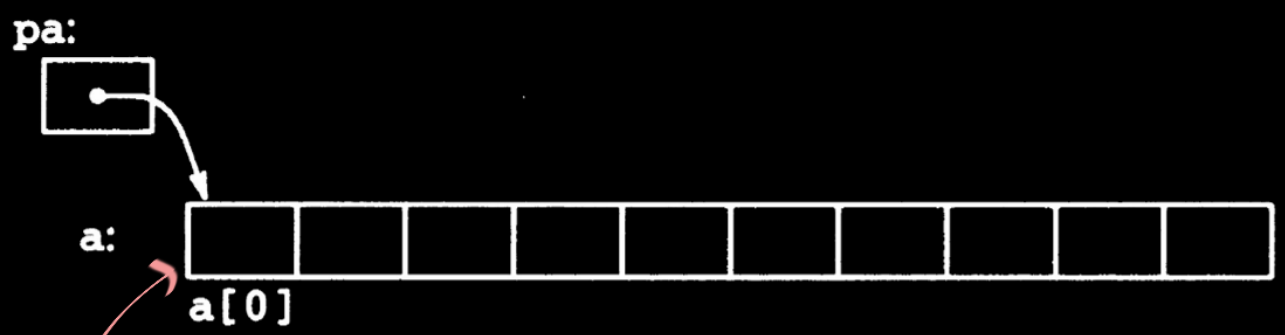
Suppose address is 64 bits
integer is 32 bits

address
↓
{ 110010...00 } 00101000
32 bits 32 bits

%p

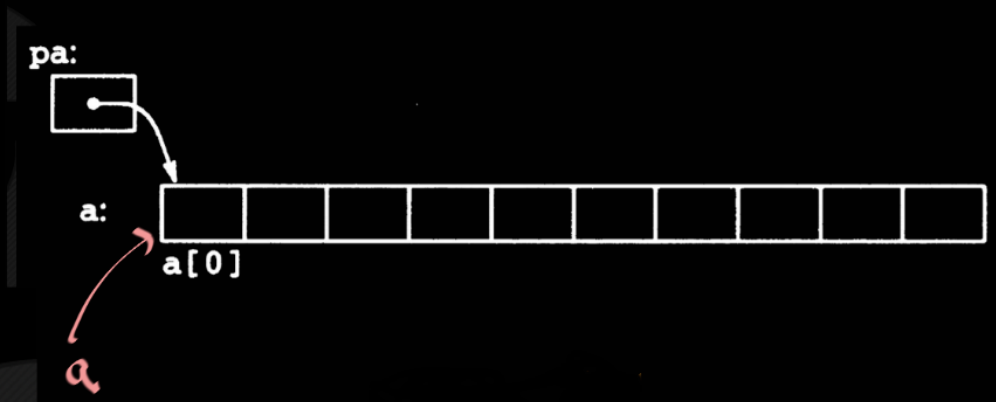
%u

```
int *pa;

int a[10];

pa = &a[0];
```



pa ≡ a ≡ &a[0]

a



Suppose Pa is also pointing to the first element of array.

What is the difference b/w Pa and a ?

→ 'a' can not be used on left hand side.

a = a+1 (illegal)

Suppose Pa is also pointing to the first element of array.

What is the difference b/w Pa and a ?

→ 'a' can not be used on left hand side.

a = a+1 (illegal)

but we can use for Pa.

→ Pa has it's own box 

, a does'nt has its own box

→ sizeof(Pa) ↳ Size of address (8 bytes) sizeof(a) ↳ 5 x 4 Bytes = 20B

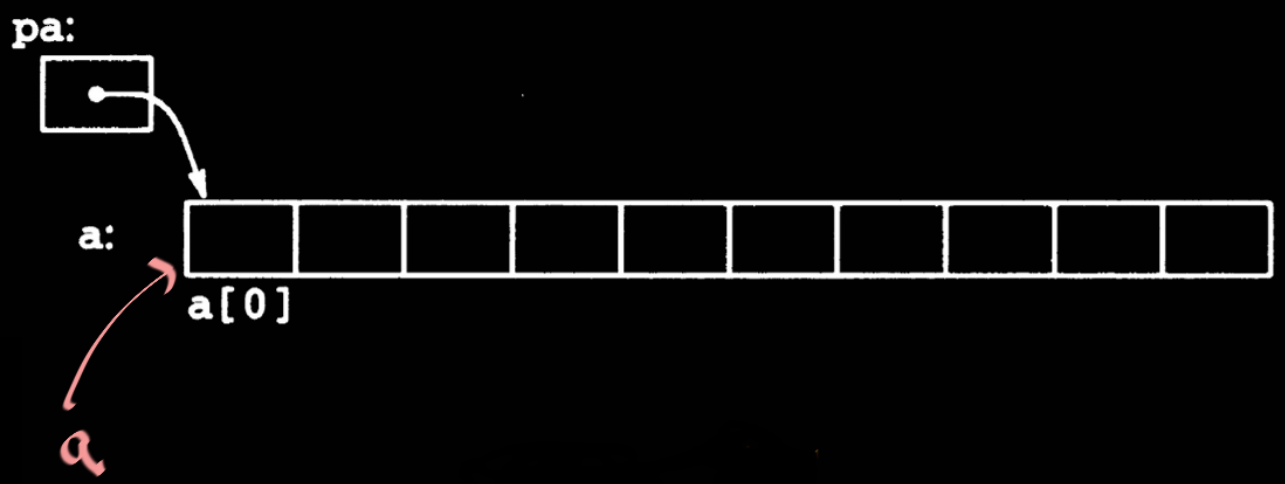
No. of integer x size of one integ

GO CLASSES

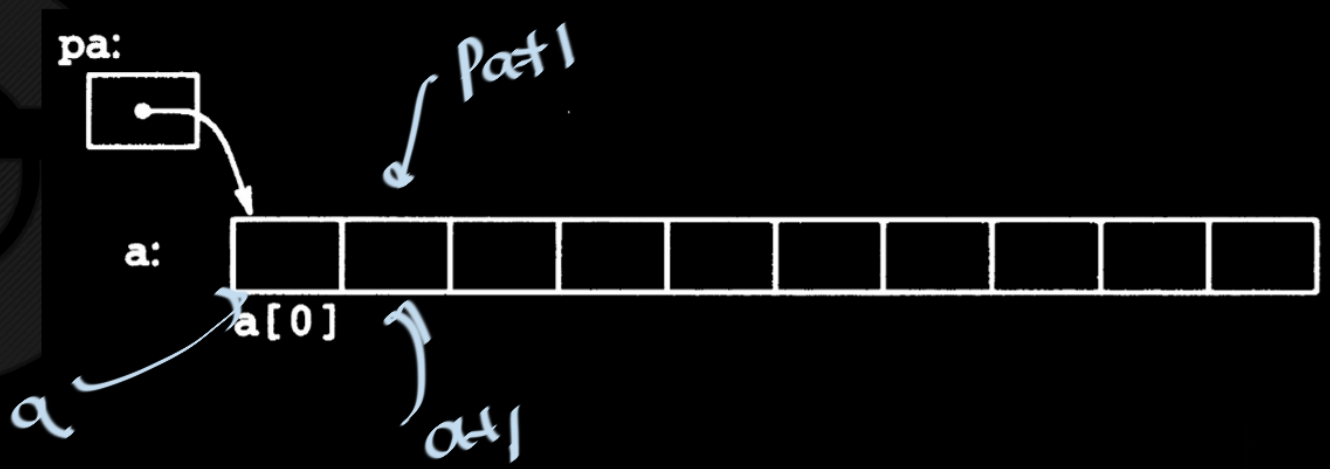
```
int *pa;

int a[10];

pa = &a[0];
```

or

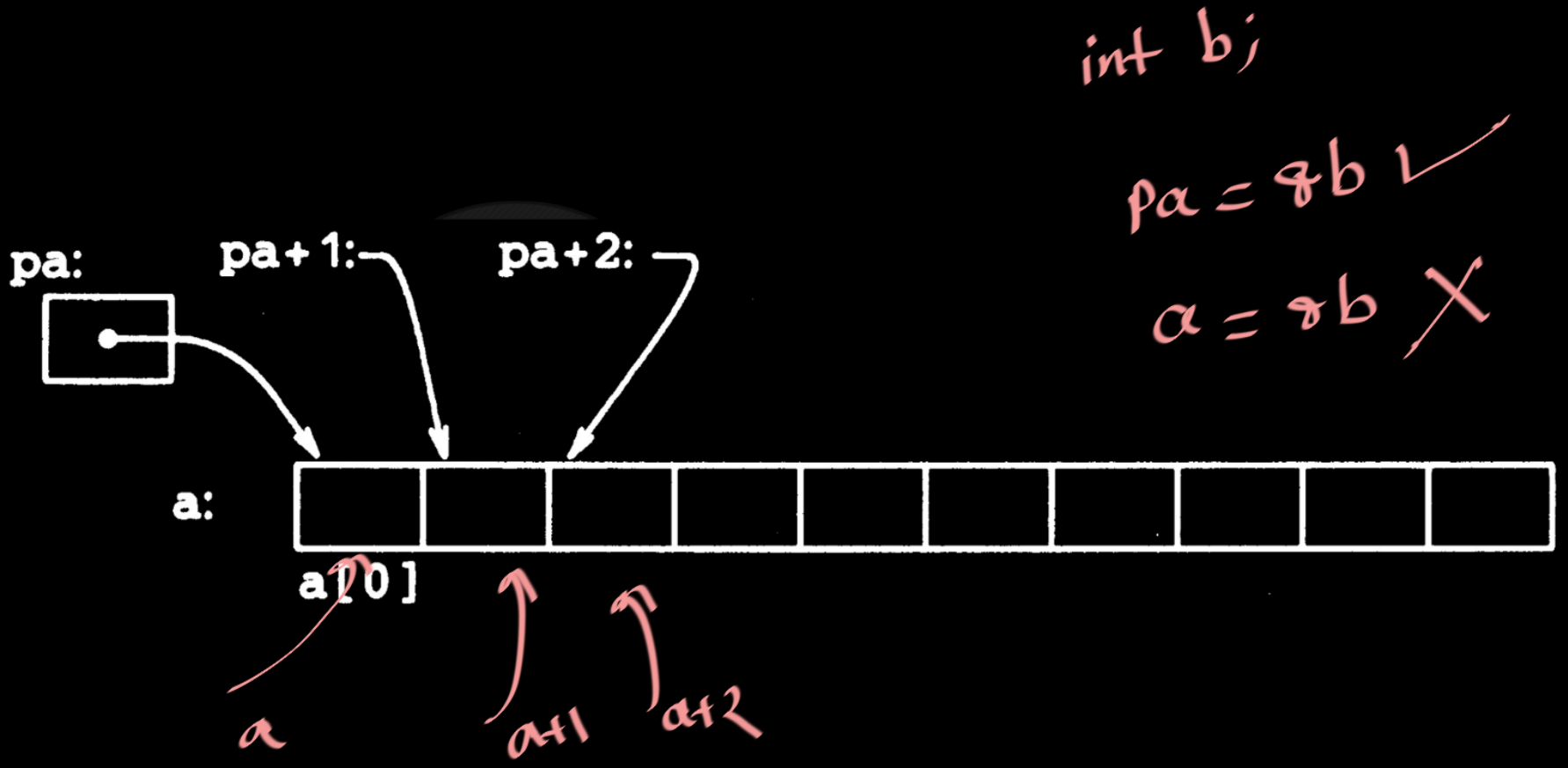
pa = a;



```
int *pa;

int a[10];

pa = &a[0];
```



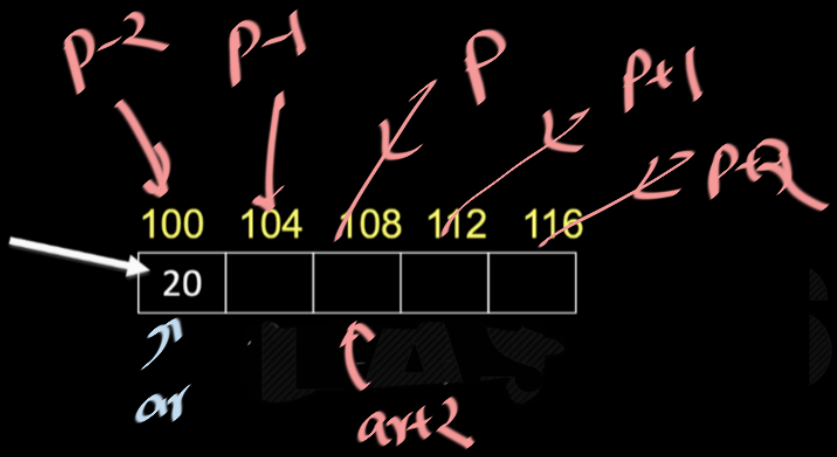
pa+1 points to ??


int b;
pa = &b ✓
a = &b ✗
pa:
pa+1:
pa+2:
a:
a[0]
a
a+1
a+2

Question 1:

Which of the assignment statements leads to compilation error ?

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

 i) p=ar+2;
 ii)ar=p+1;
```

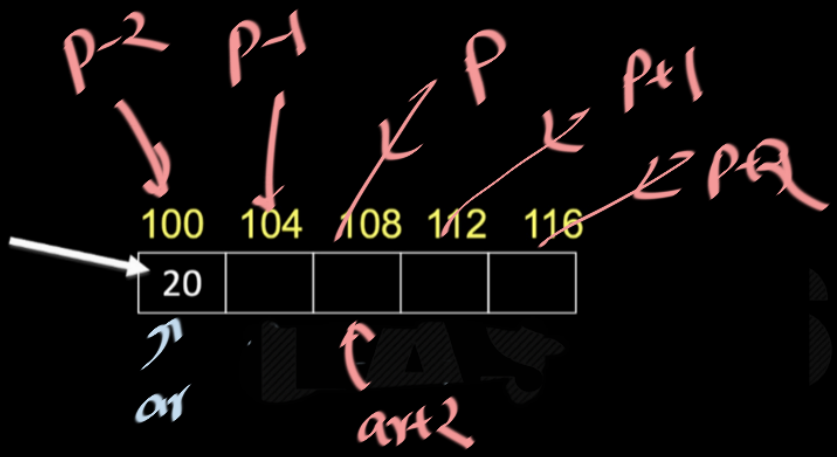


A. p=ar+2;
B. ar=p+1;
C. Both statements result in error at compile time
D. Neither results in a compilation error

Question 1:

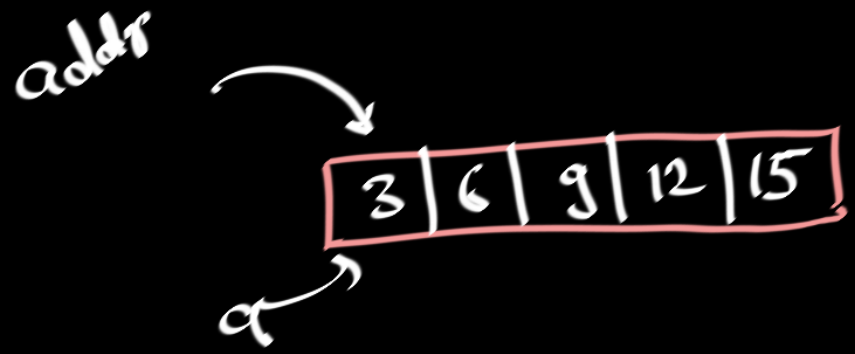
Which of the assignment statements leads to compilation error ?

```
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:




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

- What is the value of `(*addr)+4`? 7
- What is the value of `*(addr+4)`? 15

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

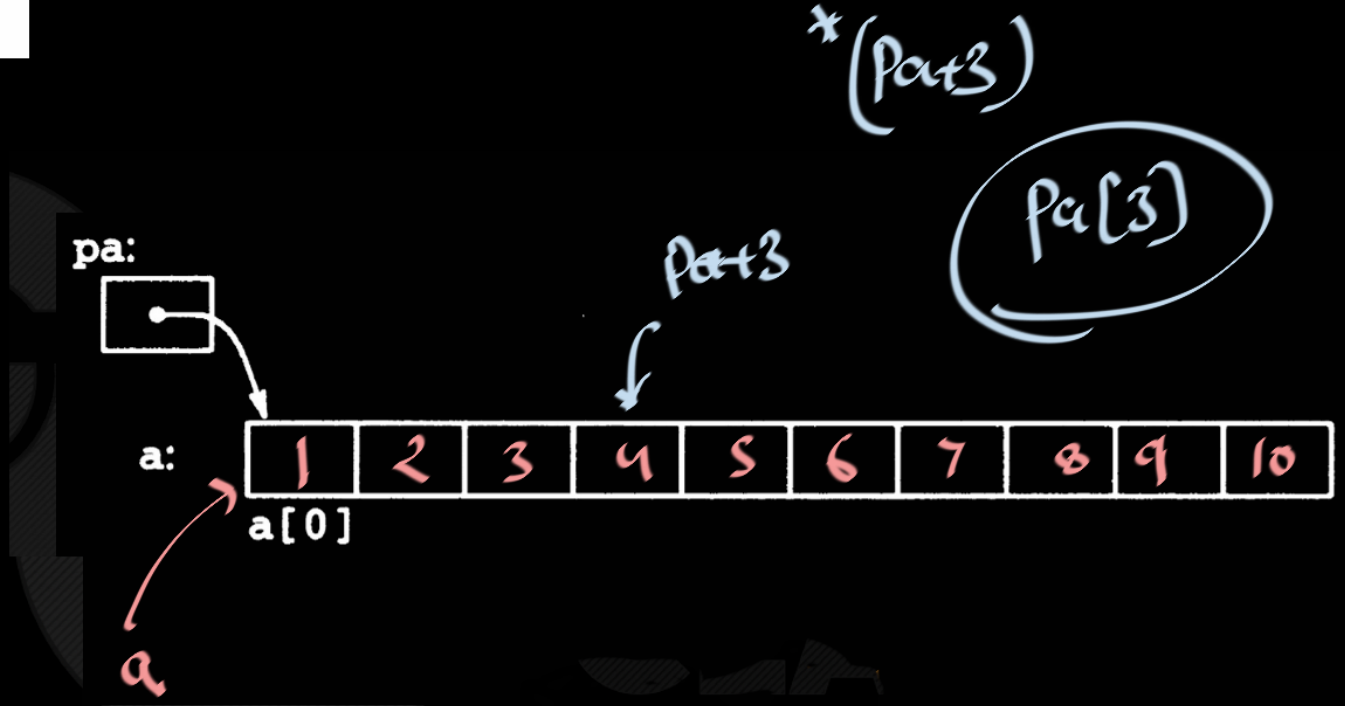
```
int* addr = &(a[0]);
```

$\neq$

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

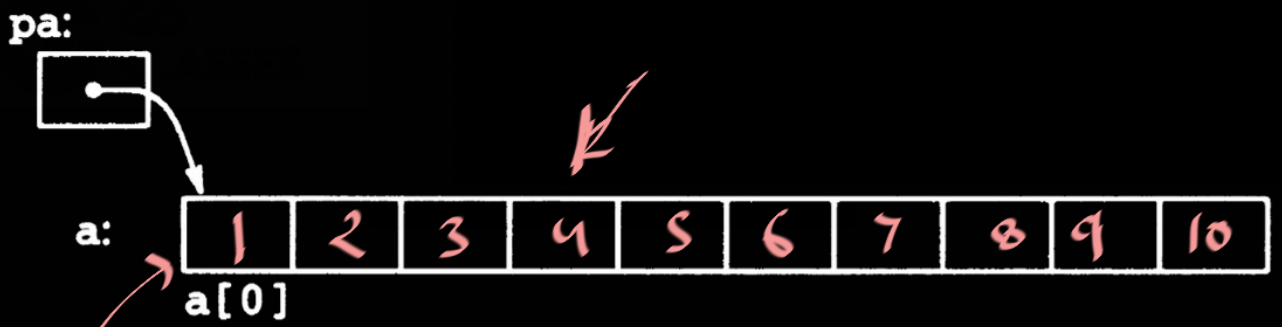
$=$

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



$a[3] \equiv *(a+3)$

$3[a] \equiv (3+a)^*$



6 Ways to access a[3]

1) a[3]

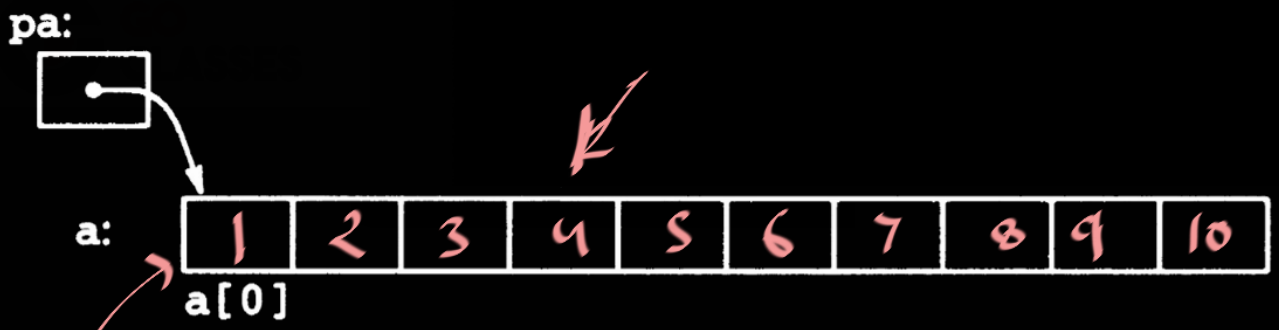
2) 3[a]

3) *(a+3)

4) pa[3]

5) 3[pa]

6) *(pa+3)



1) a[3]

2) 3[a]

3) *(a+3)

4) pa[3]

5) 3[pa]

6) *(pa+3)

Preferred ways

Question 3:

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

which of the following statements is ***NOT TRUE***

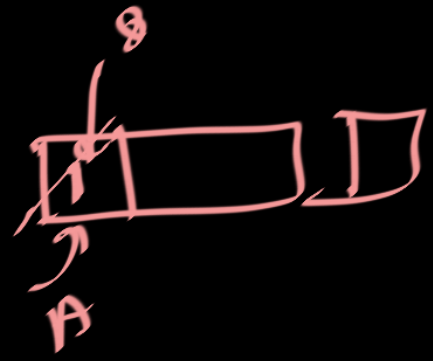
[a.] `p = A;` is equivalent to `p = &A[0];` T

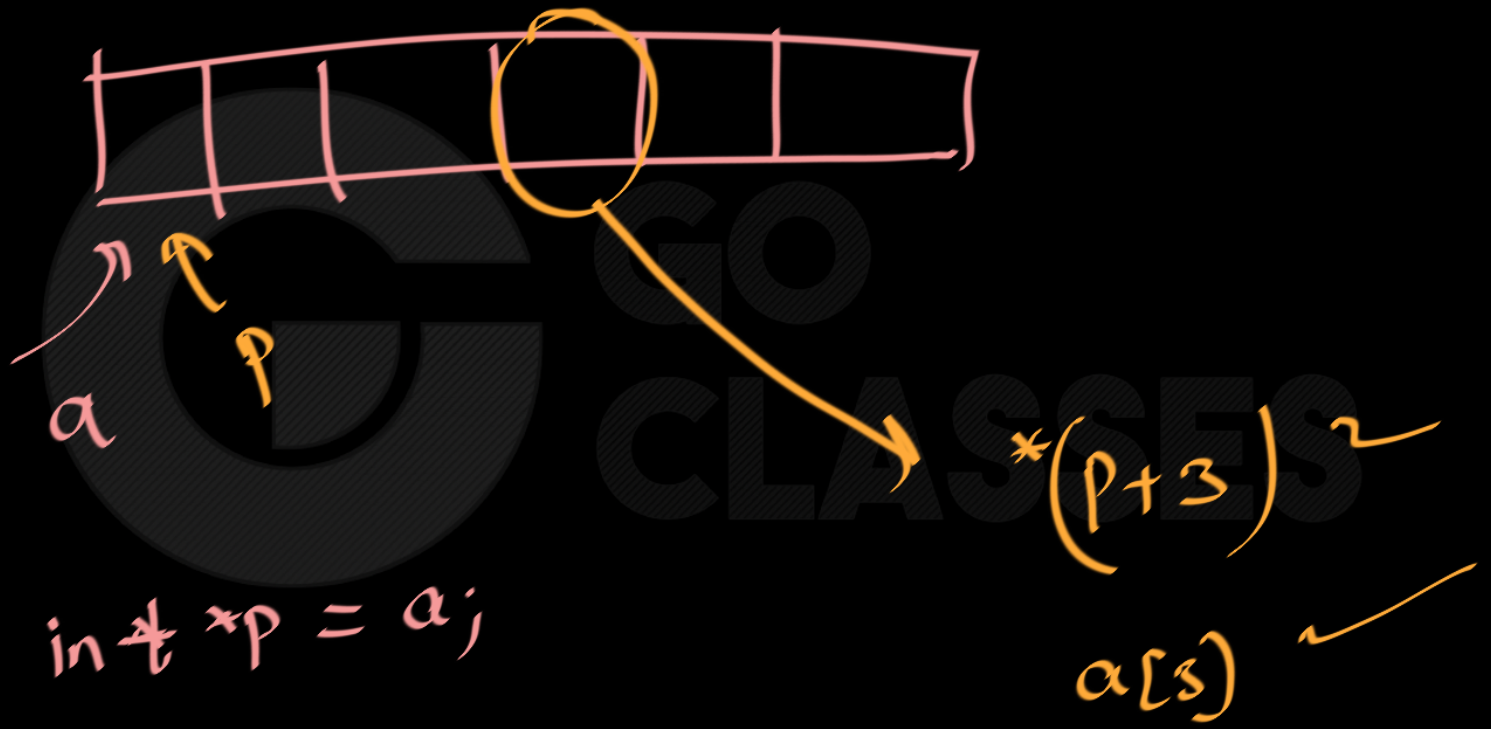
[b.] `x = p[ 4 ];` is equivalent to `x = *(p + 4);` T

[c.] `x = p[ 3 ];` is equivalent to `x = *(A + 3);` T

[d.] `x = *A + 7;` is equivalent to `x = A[ 7 ];` F

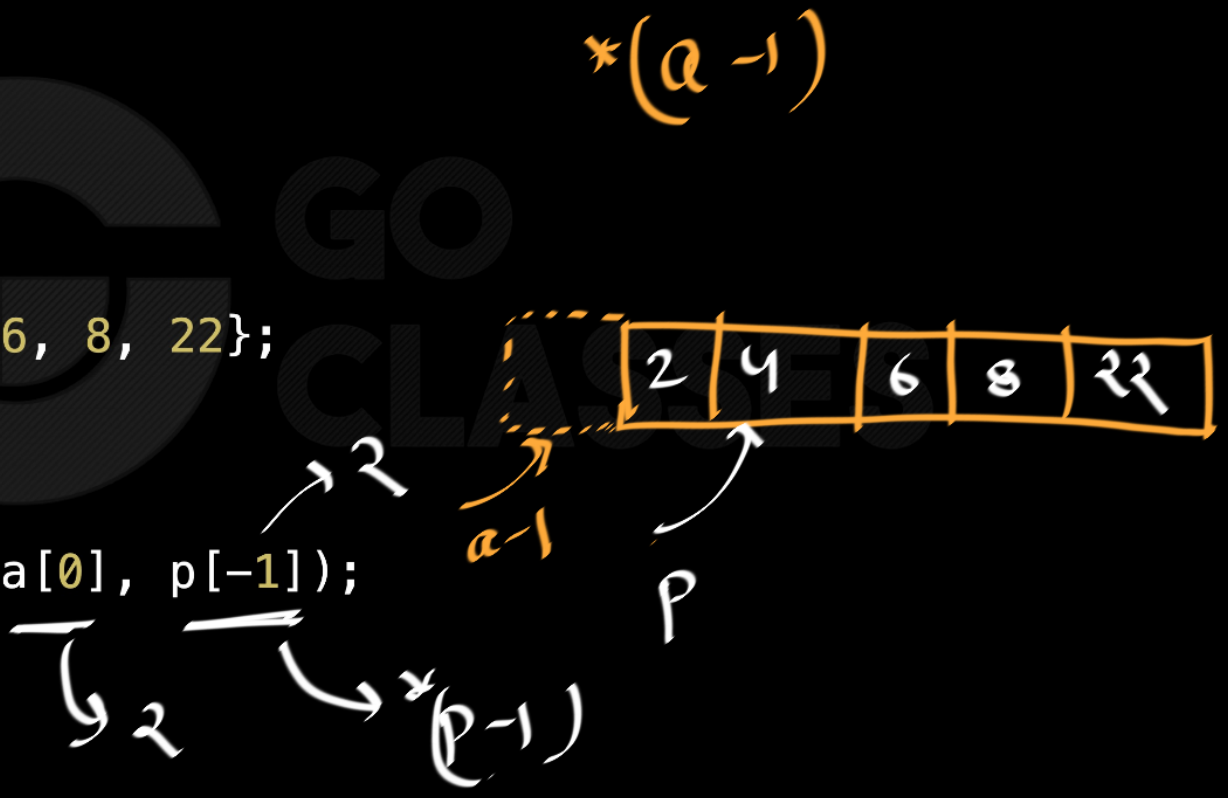
↓ *(A+7)




a
p
*(p+3)
int *p = a;
a[3]

Question 4:

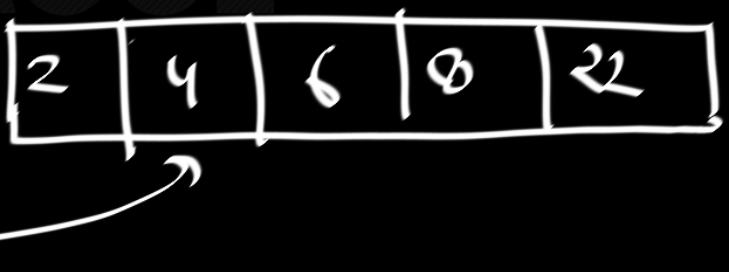
```c
#include <stdio.h>

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;
}
```



GO CLASSES

```c
#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;
```

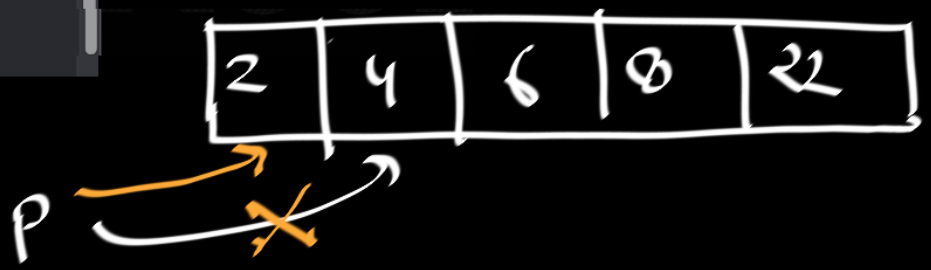
```
Desktop — -zsh — 63×10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c -
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
2 2
4
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

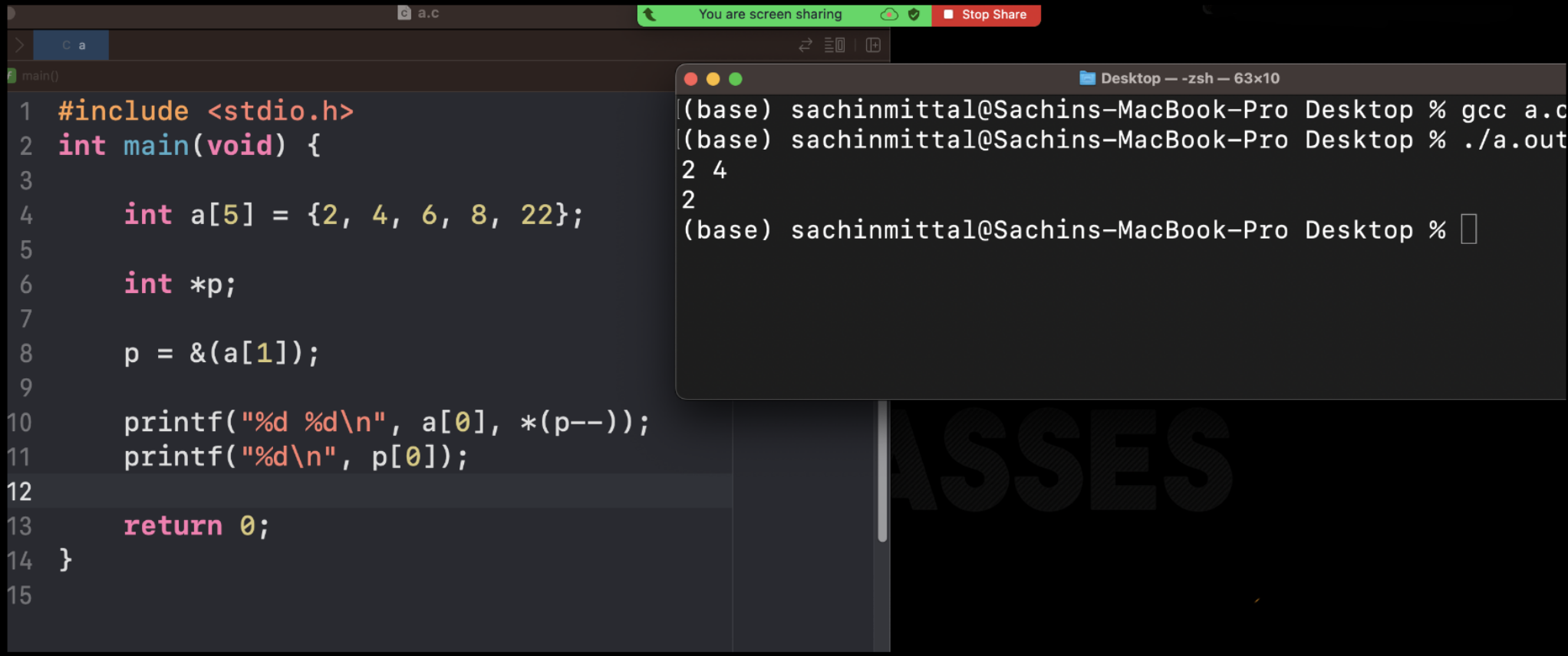


GO CLASSES

```c
#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 — 63×10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
2 2
2
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```




a.c
You are screen sharing
Stop Share
main()
#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 — 63×10
(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 %

GO CLASSES

* (a-1)

a.c

```c
#include <stdio.h>
int main(void) {

    int a[5] = {2, 4, 6, 8, 22};


    printf("%d\n", a[-1]);

    return 0;
}
```

Desktop — -zsh — 63×10

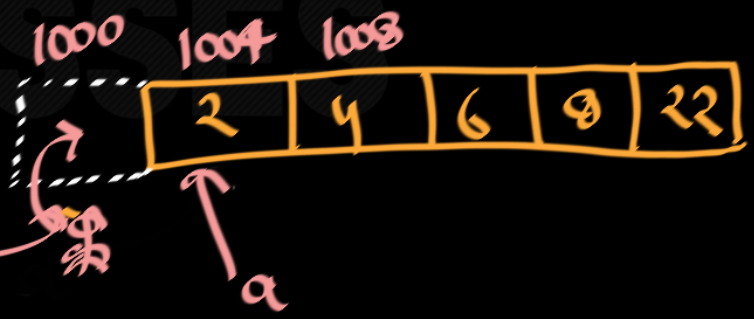
```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c -w
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
0
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

Run time error

a-1 ≡ 1000

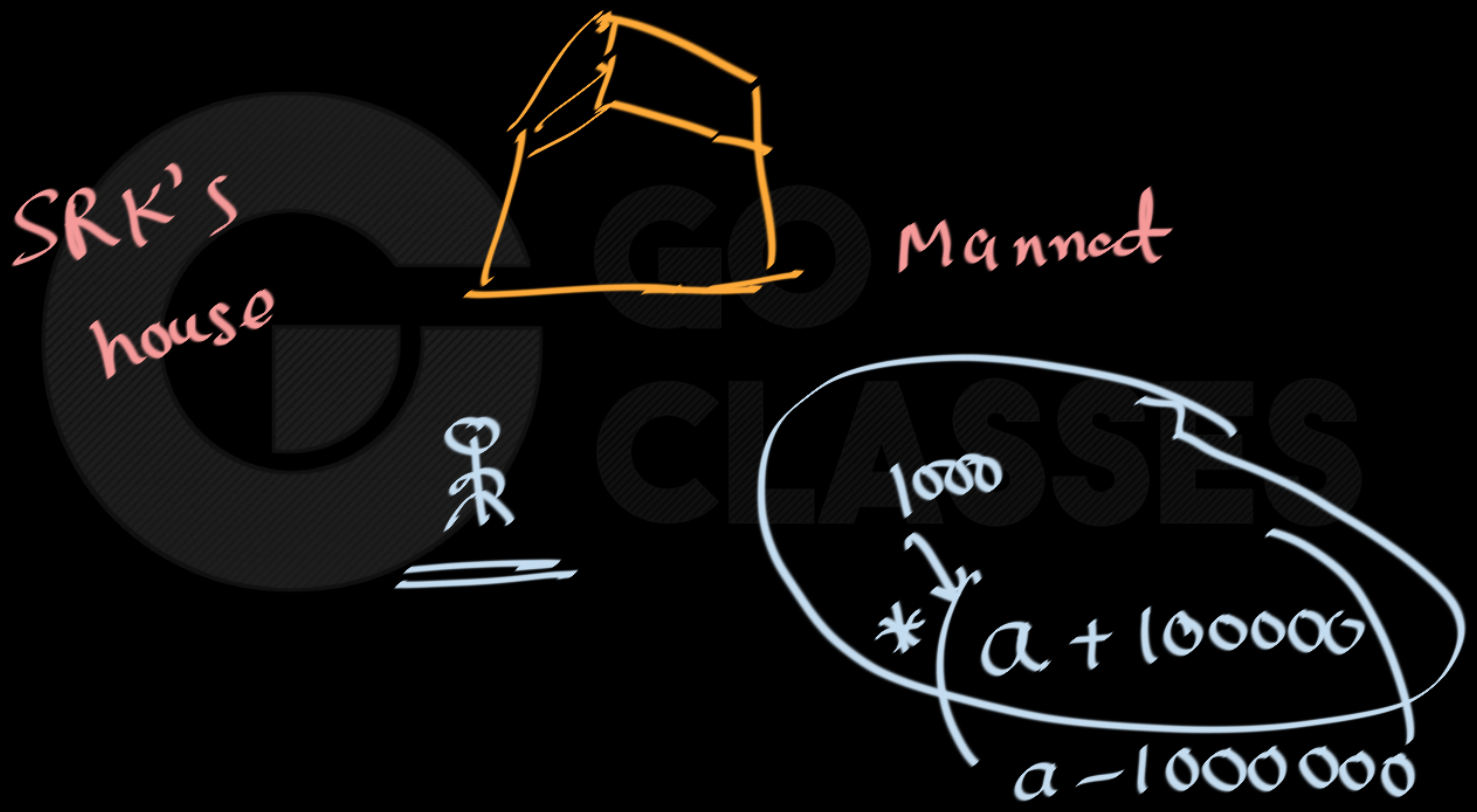
* (a-1)

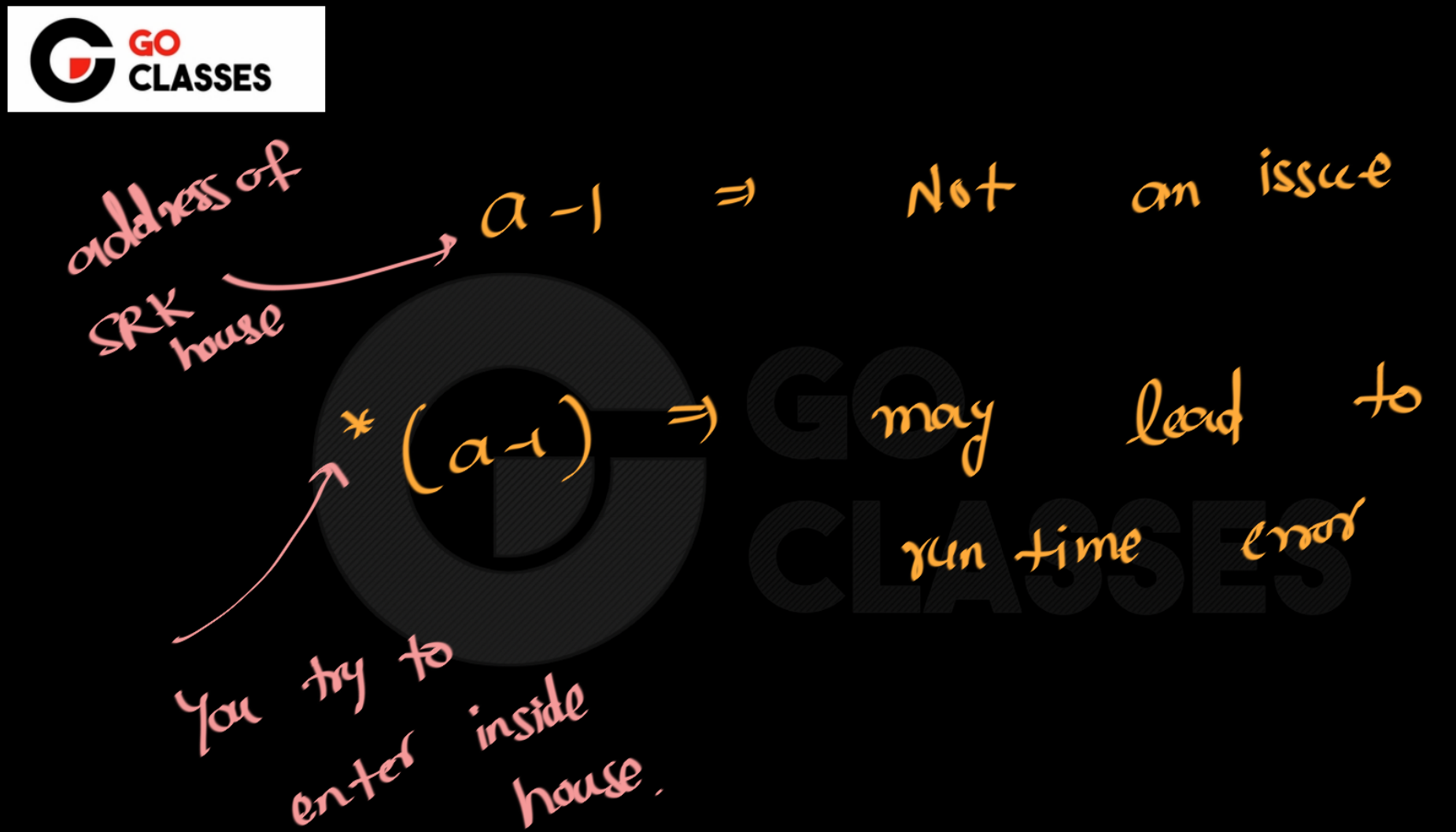
a-1



GO CLASSES

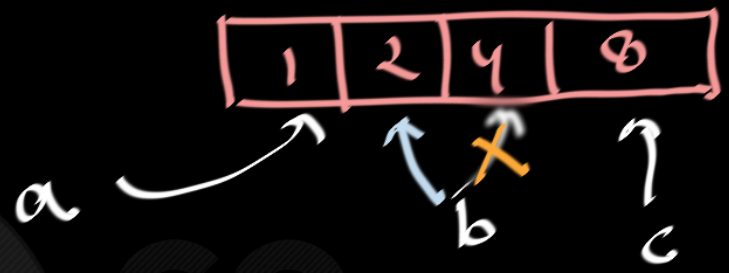
- Run time error
  (segmentation fault)
  - these errors occurs when we try to read / write into the unknown memory


SRK's house
Manned
1000
*
a + 100000
a - 1000000


address of SRK house
a - 1
⇒
Not an issue
* (a - 1)
⇒
may lead to run time error
You try to enter inside house.

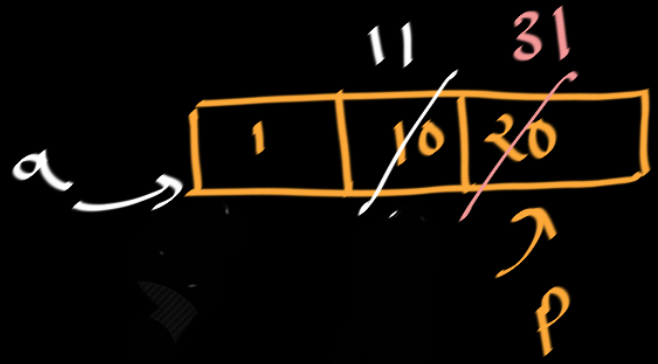
Question 5:

```c
#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

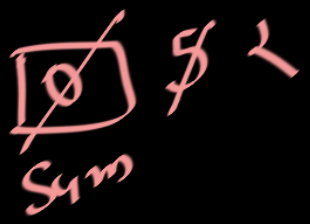
Question 6:

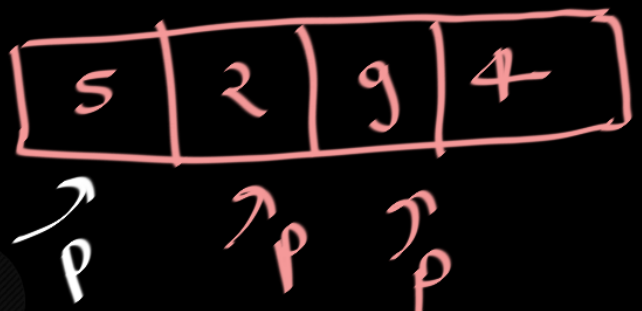



```
int a[] = {1,10,20}, *p=a, i=1, j=2;
for (i=0, p=a; i < 2; i++) {
  int j = *p;  p++;  *p = *p + j;
}
printf("out: i=%d j=%d a=%d,%d,%d\n", i, j, a[0], a[1], a[2]);
```

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;
}
```





p = a

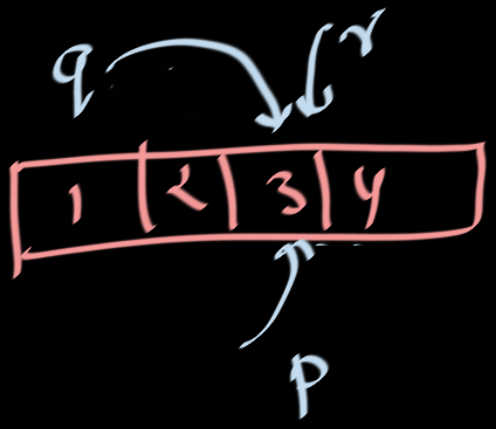
p = a+1

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;
```



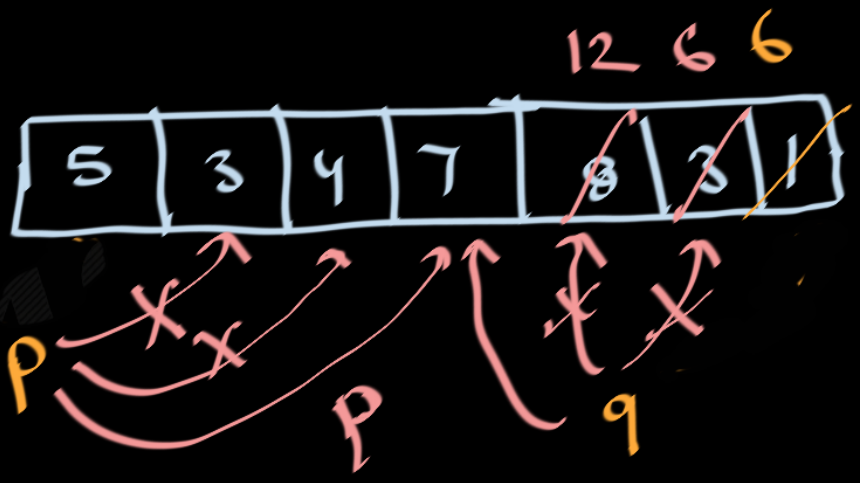

```
printf("%d \n", *(p + 1));
printf("%d \n", *(p - 1));
printf("%d \n", *q);
printf("%d \n", *r);
```

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;
   p++;
   q--;
}
```

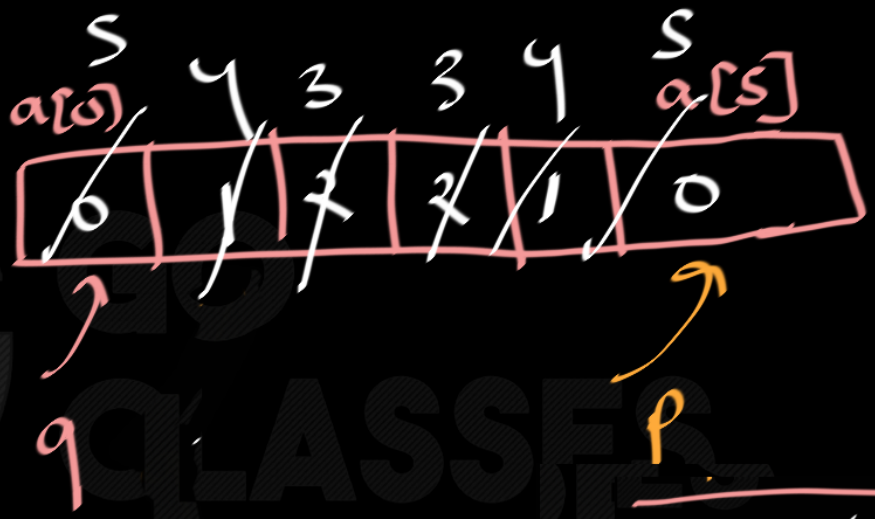
*q += *p; ⟶ *q = *q + *p



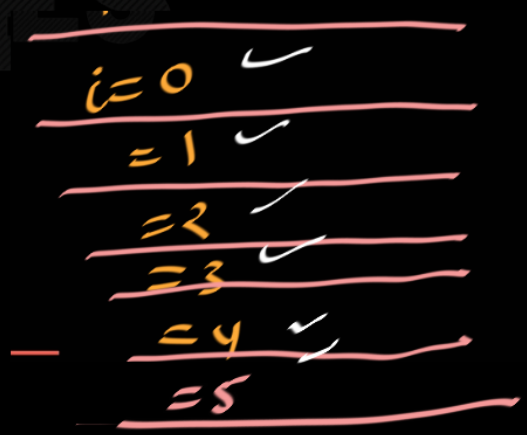
a[0]: 5 a[1]: 3 a[2]: 4 a[3]: 7 a[4]: 12 a[5]: 6 a[6]: 6

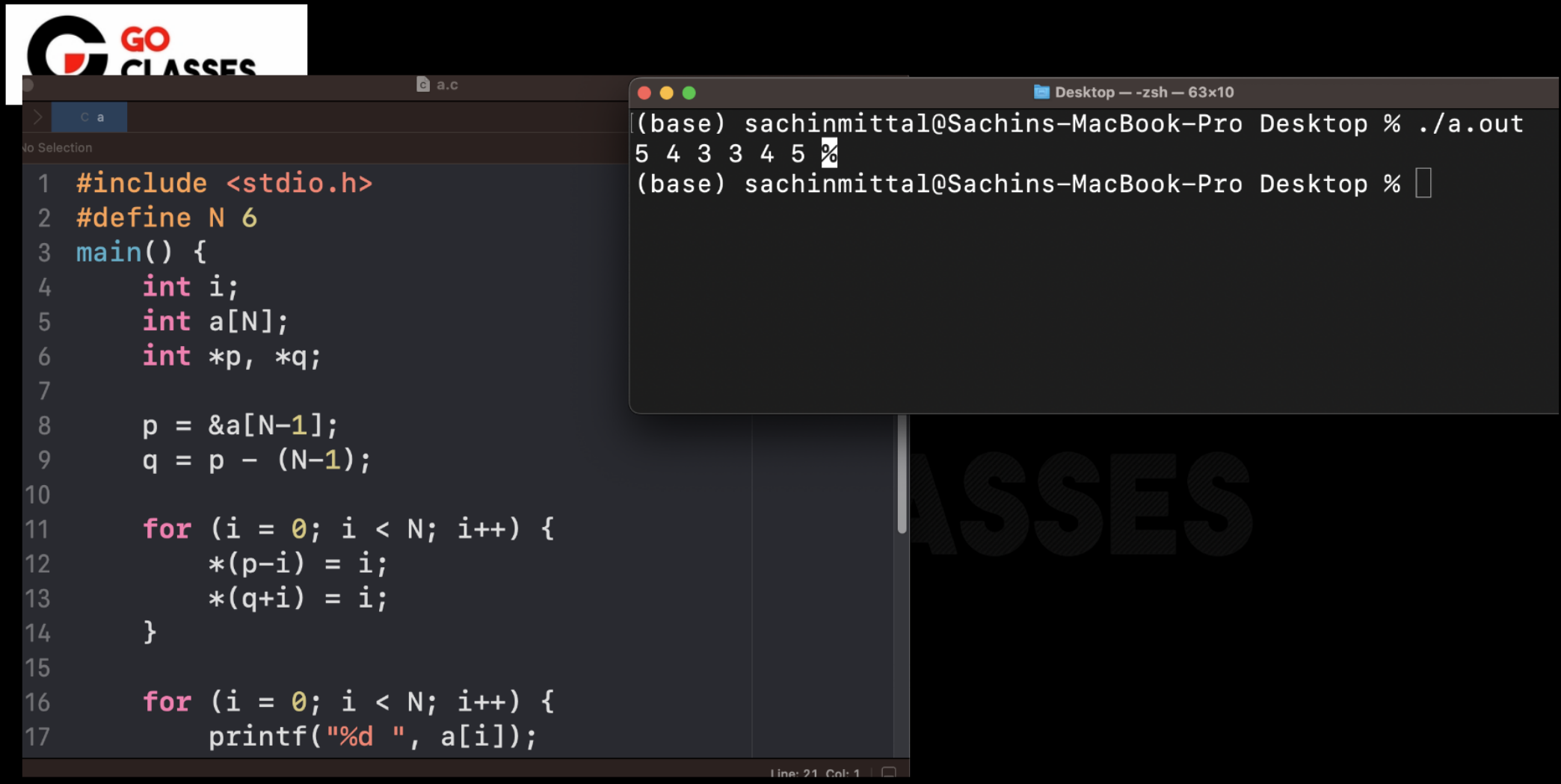
Question 10: What does the following program print?

```
#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]);
    }

}
```



5 4 3 3 4 5

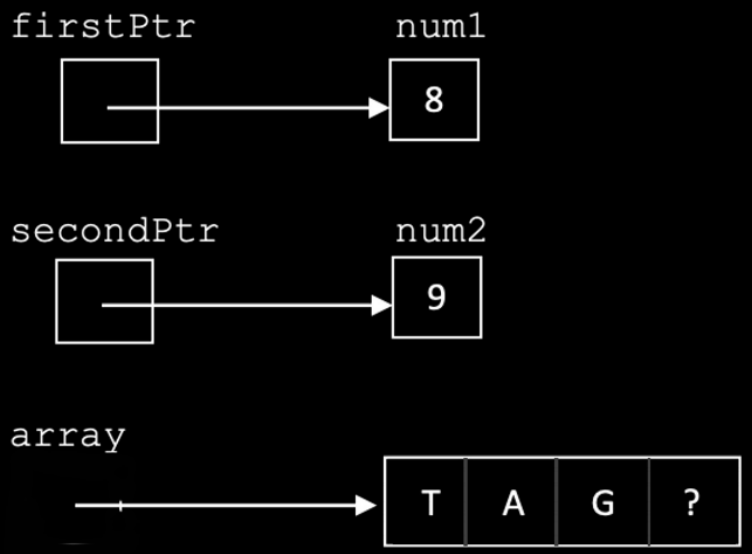



a.c
No Selection
1 #include <stdio.h>
2 #define N 6
3 main() {
4 int i;
5 int a[N];
6 int *p, *q;
7
8 p = &a[N-1];
9 q = p - (N-1);
10
11 for (i = 0; i < N; i++) {
12 *(p-i) = i;
13 *(q+i) = i;
14 }
15
16 for (i = 0; i < N; i++) {
17 printf("%d ", a[i]);
Desktop — -zsh — 63×10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
5 4 3 3 4 5 %
(base) sachinmittal@Sachins-MacBook-Pro Desktop %

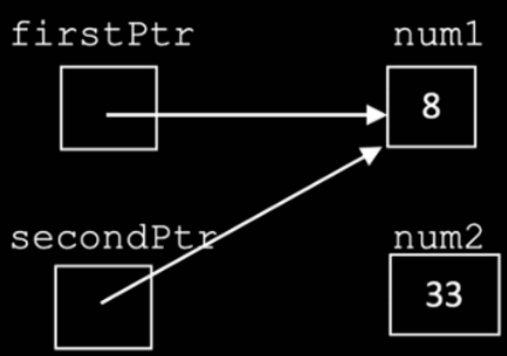
Question 11:

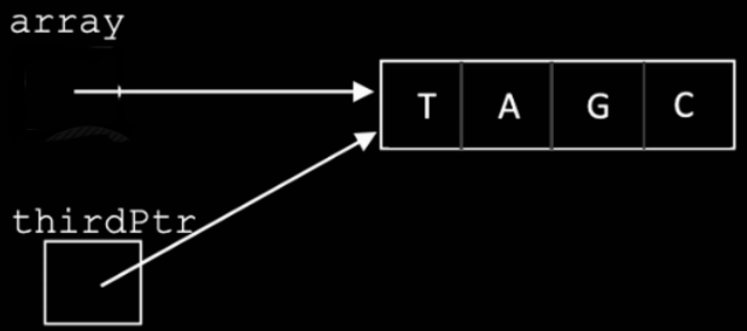
The following code creates structures represented by the diagrams below:

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

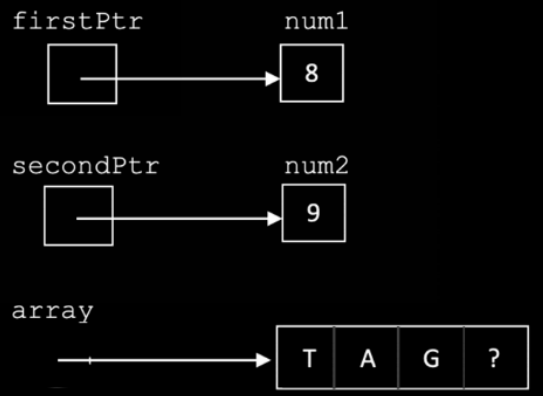


Write the few lines of code necessary to change the above diagrams to the diagrams below, but do not use the variable names `num1` and `num2` in your solution.





initial



Answer:

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

GO CLASSES

Pointer Arithmetic
and
sizeof operator

sizeof operator

sizeof operator

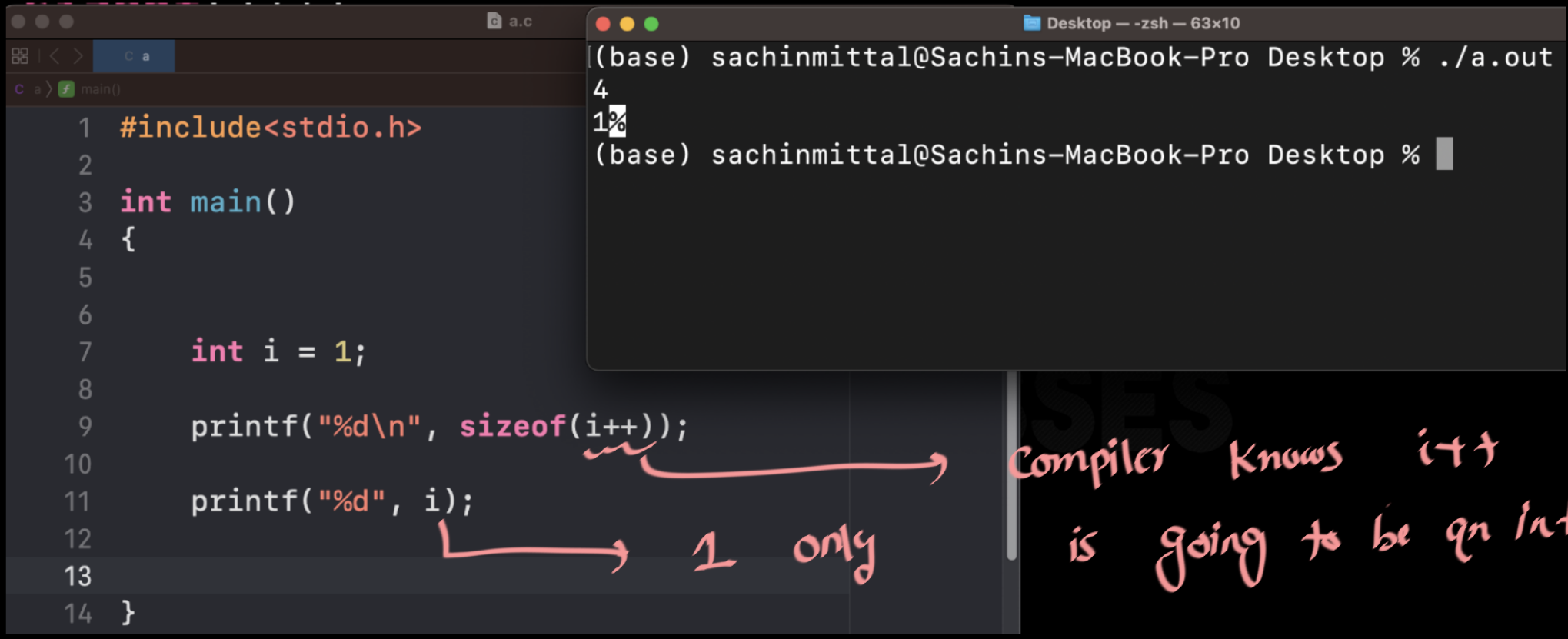
(+ , − , ++, -- , /)

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

```c
#include<stdio.h>
int main()
{

    printf("%d", sizeof(int));

}
```

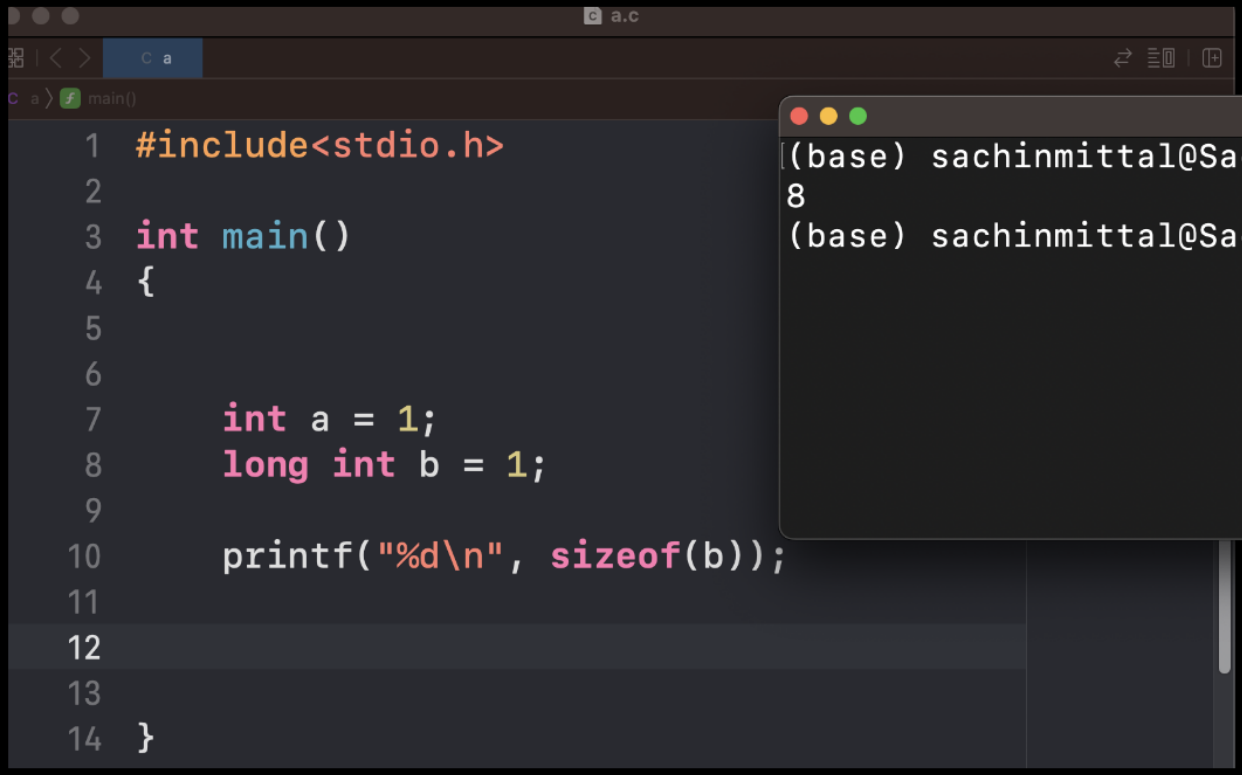
```c
#include<stdio.h>
int main()
{
    int a;
    printf("%d", sizeof(a));

}
```

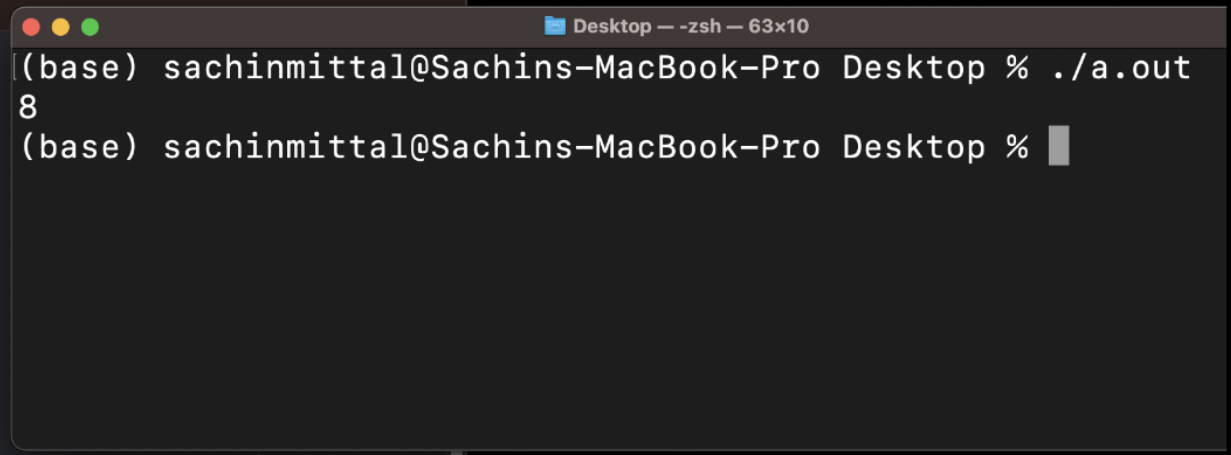
sizeof(i++)


a.c
main()
1 #include<stdio.h>
2
3 int main()
4 {
5
6
7 int i = 1;
8
9 printf("%d\n", sizeof(i++));
10
11 printf("%d", i);
12
13
14 }
Desktop — -zsh — 63×10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
4
1%
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
1 only
Compiler knows itt
is going to be an int

Compiler doesn't evaluate any expression inside sizeof operator.

GO CLASSES


a.c
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 }


Desktop — -zsh — 63×10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
8
(base) sachinmittal@Sachins-MacBook-Pro Desktop %

sizeof(Expression)

Example :-

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

GO CLASSES

a.c

```c
#include<stdio.h>

int main()
{


    int a = 1;
    long int b = 1;

    printf("%d\n", sizeof(a+b));



}
```

Desktop — -zsh — 63×10

```
(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 CLASSES

a.c

```c
#include<stdio.h>

int main()
{

    char c = 'a';
    int a = 1;
    long int b = 1;

    printf("%d\n", sizeof(c+a));



}

```

Desktop — -zsh — 63×10

```
(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 %
```

GO CLASSES

optional (integer Promotion)

a.c

```c
#include<stdio.h>

int main()
{

    char c = 'a';
    char d = 'b';

    int a = 1;
    long int b = 1;

    printf("%d\n", sizeof(c+d));



}
```

Desktop — -zsh — 63×10

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

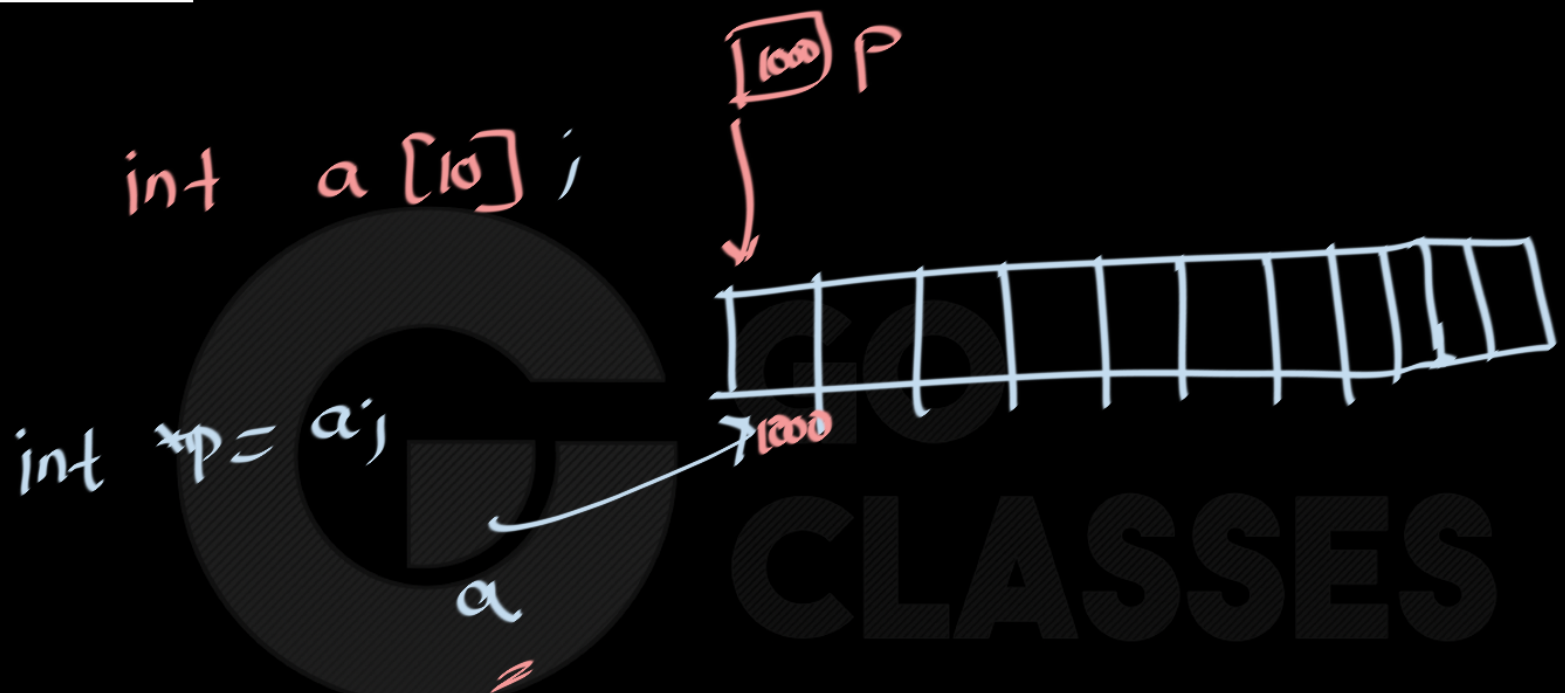
sizeof(array_name)

a.c

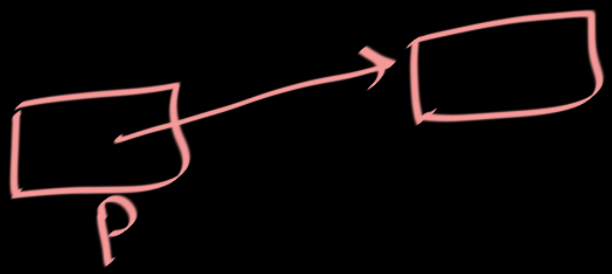
```c
#include<stdio.h>

int main()
{

    int a[10];

    printf("%d\n", sizeof(a));



}
```

```
Desktop — -zsh — 63×10
(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] i
1000
P
int *p = a;
1000
a
2

`sizeof(address)` or `sizeof(pointer)`



sizeof()

long int *li;

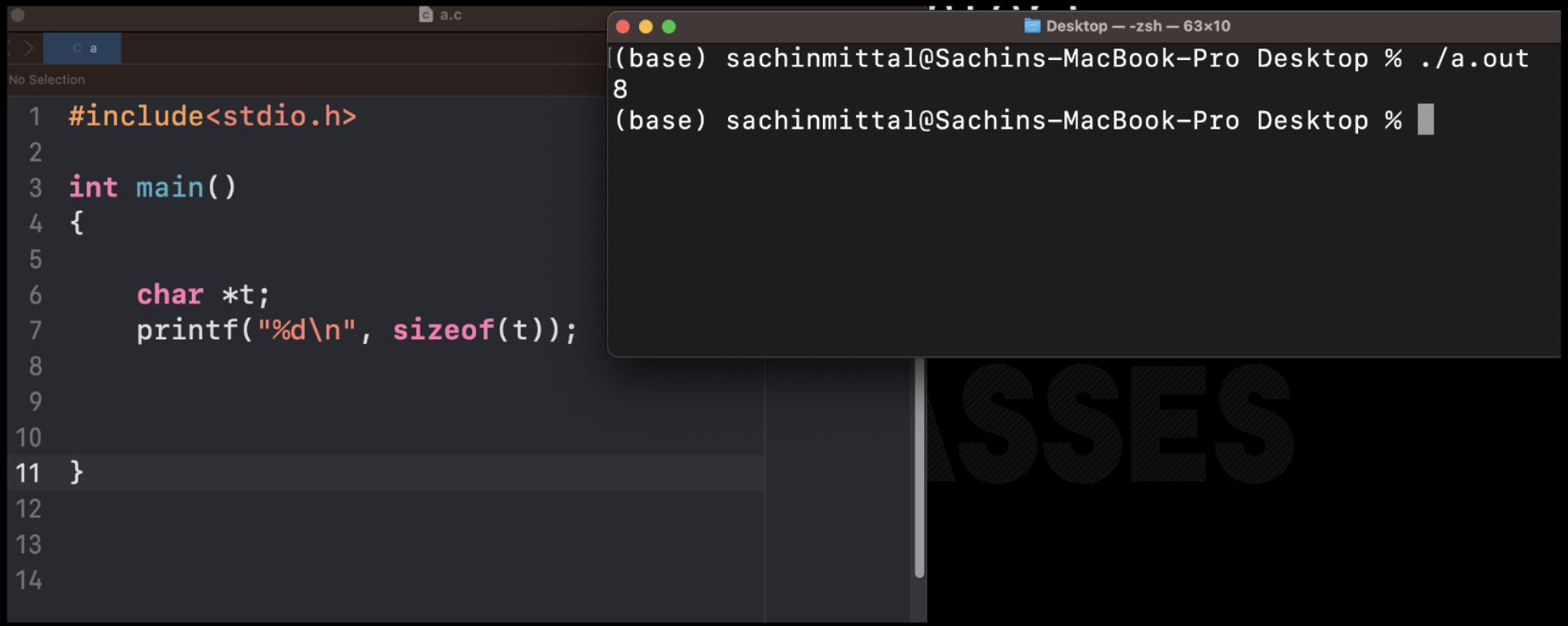
int *a;

char *ch;

float *f;

short int *st;

double *d;


a.c
No Selection
#include<stdio.h>
int main()
{
char *t;
printf("%d\n", sizeof(t));
}
Desktop — -zsh — 63×10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
8
(base) sachinmittal@Sachins-MacBook-Pro Desktop %

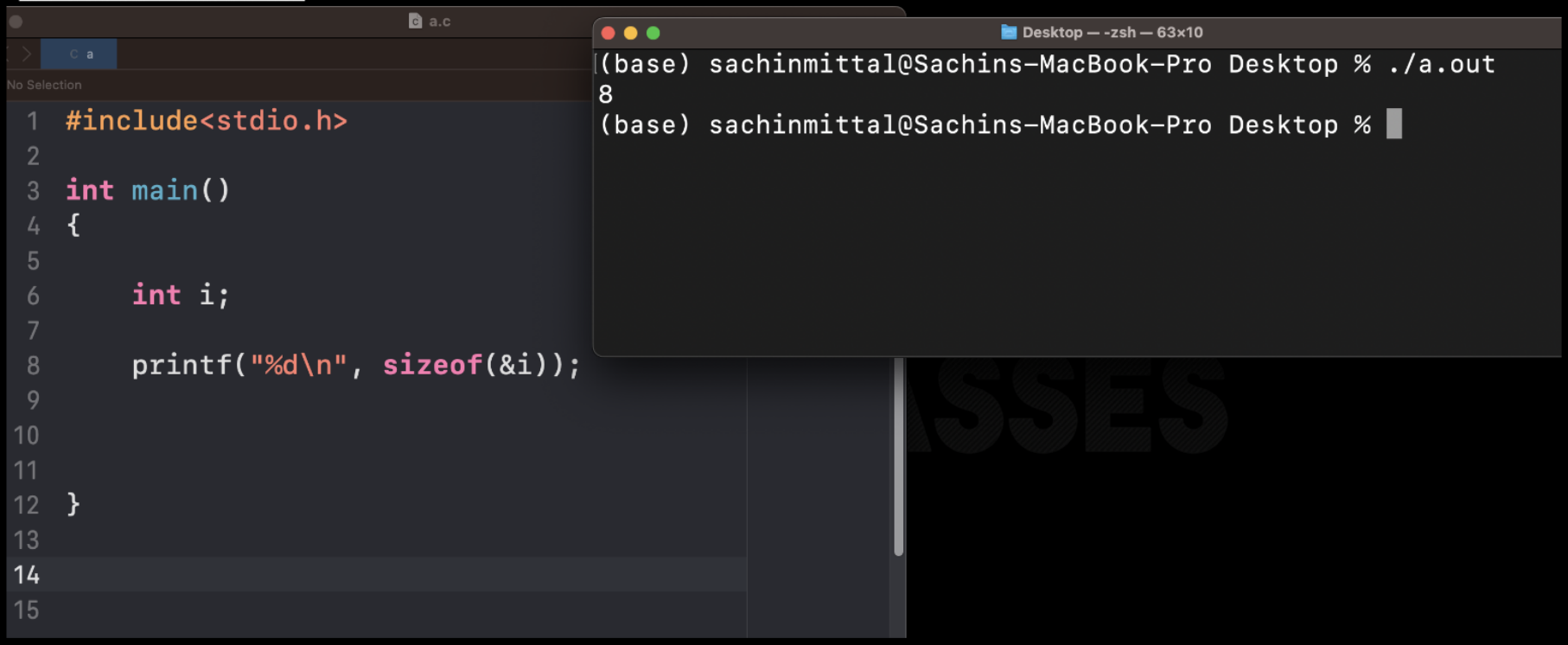
GO CLASSES

a.c

```c
#include<stdio.h>

int main()
{

    char *t;
    char c;

    printf("%d\n", sizeof(&c));



}
```

Desktop — -zsh — 63×10

```
(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
#include<stdio.h>
int main()
{
int i;
printf("%d\n", sizeof(&i));
}
Desktop — -zsh — 63×10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
8
(base) sachinmittal@Sachins-MacBook-Pro Desktop %

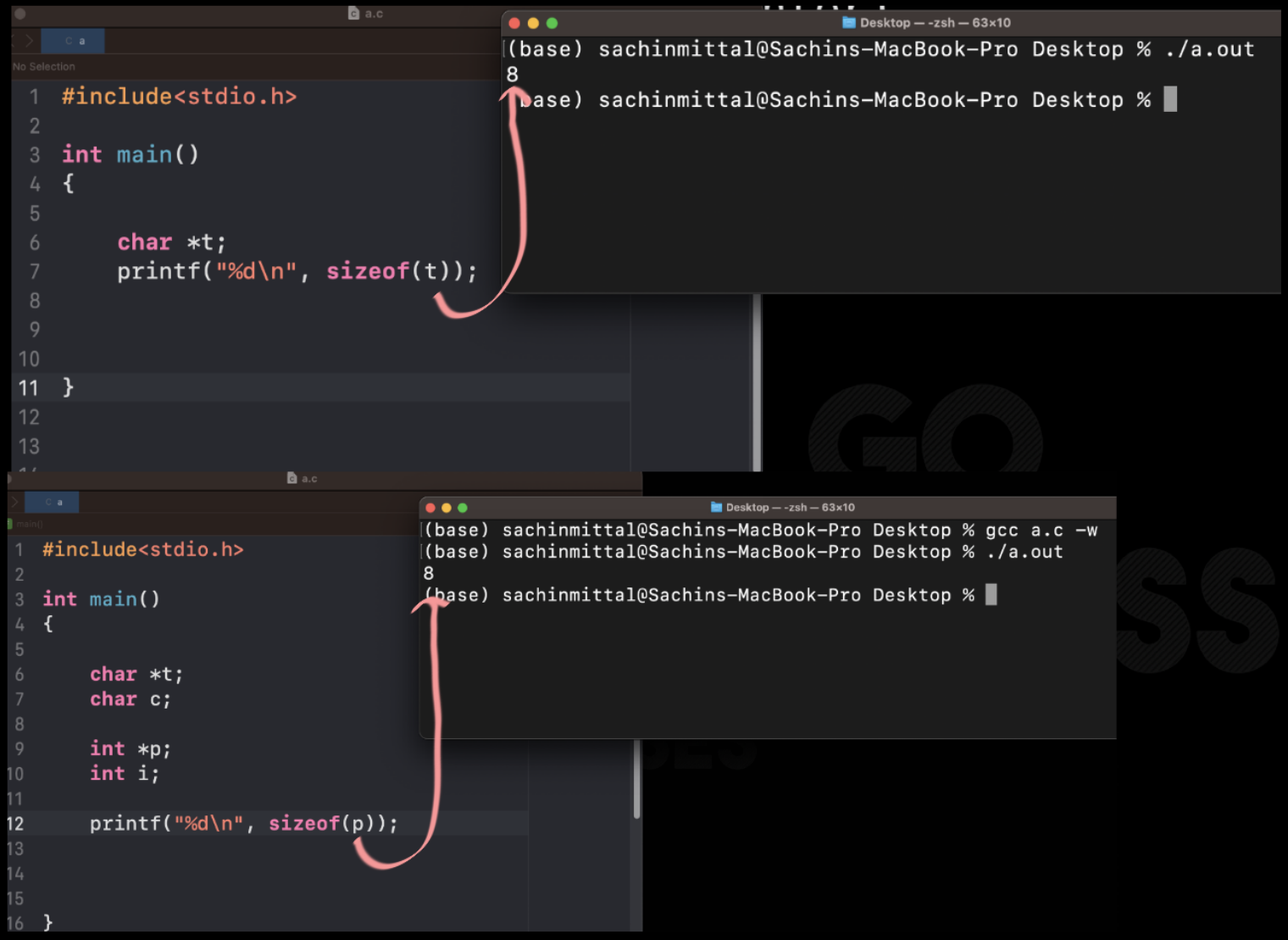
GO CLASSES

a.c

```c
#include<stdio.h>

int main()
{

    char *t;
    char c;

    int *p;
    int i;

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



}
```

Desktop — -zsh — 63×10

```
(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
#include<stdio.h>
int main()
{
char *t;
printf("%d\n", sizeof(t));
}
Desktop — -zsh — 63×10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
8
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
a.c
#include<stdio.h>
int main()
{
char *t;
char c;
int *p;
int i;
printf("%d\n", sizeof(p));
}
Desktop — -zsh — 63×10
(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 %

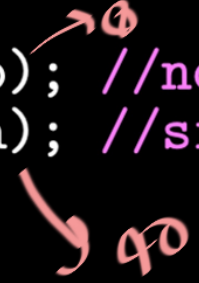
```c
int a[5];

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

4

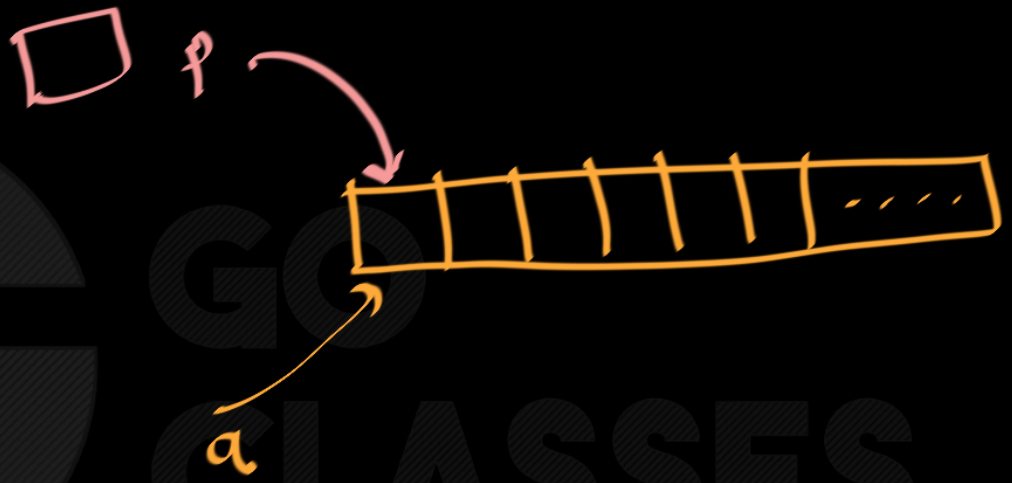
4

5 x 4 = 20

```
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)
```



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

p = a;
```



GO CLASSES

a.c

```c
#include<stdio.h>

int main()
{

    int a[10];
    int *p;

    p = a;


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


}
```

Desktop — -zsh — 63×10

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
8
40
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```


Pointer
III
address

Pointer Arithmetic

Pointer Arithmetic

`(a+i)` is the pointer arithmetic

`i=1` results in the following "assembler code":

```
a + sizeof(one element)
```

a+1

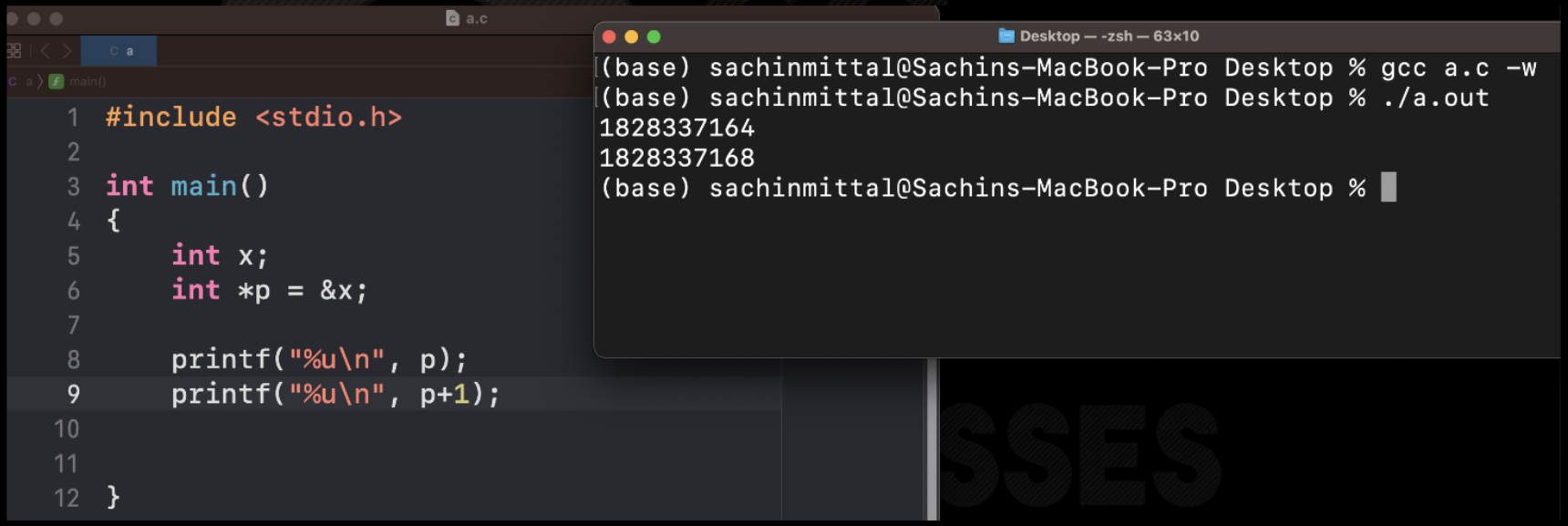
Otherwise the "assembler code" is:

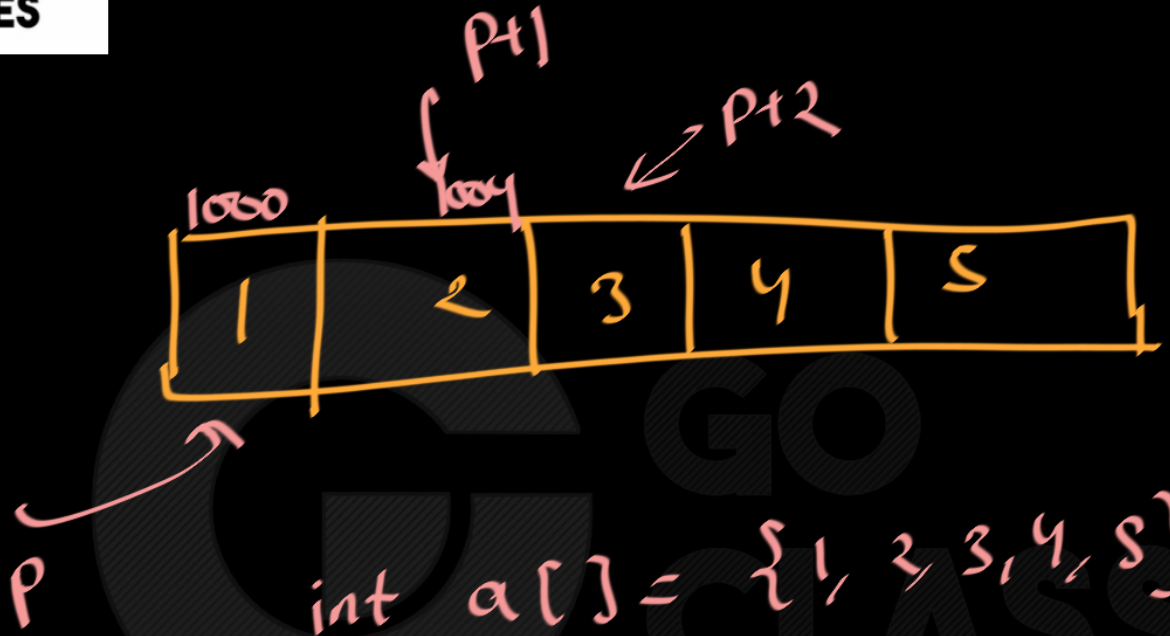
```
a + i * sizeof(one element)
```

GO CLASSES

int x;

int *p = &x;

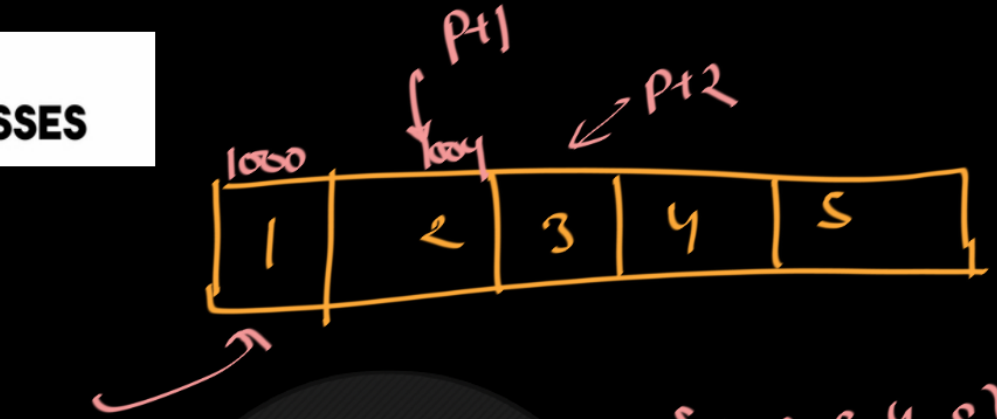





```
int a[] = {1, 2, 3, 4, 5}
int *p = a;
```

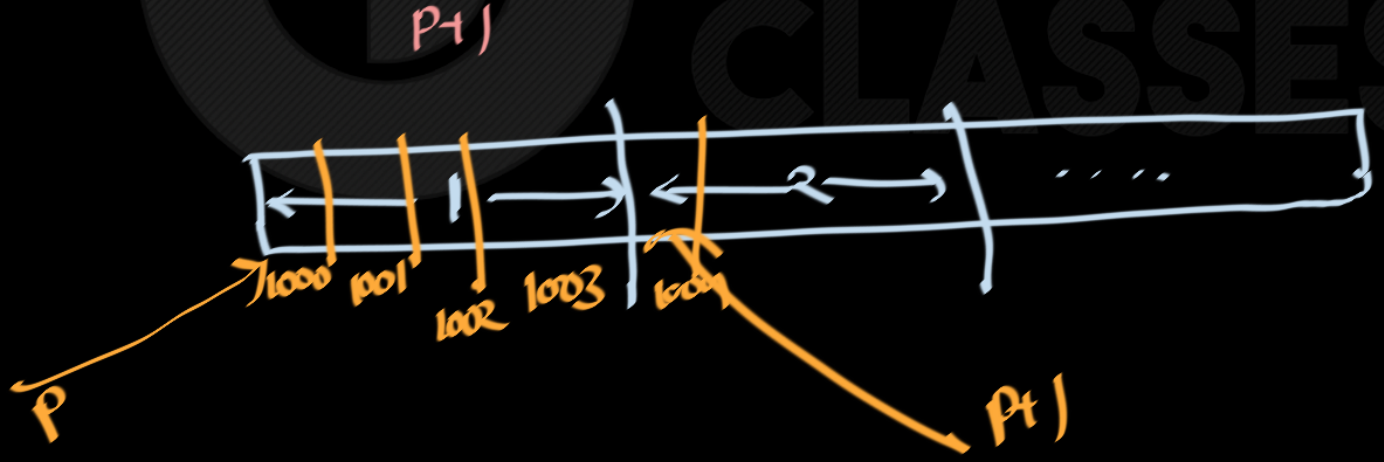
p ≡ 1000

p+1




```
int a[] = {1, 2, 3, 4, 5}
int *p = a;
```

P ≡ 1000



GO CLASSES

a.c

```c
#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));

}
```

Desktop — -zsh — 63×10

```
(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
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

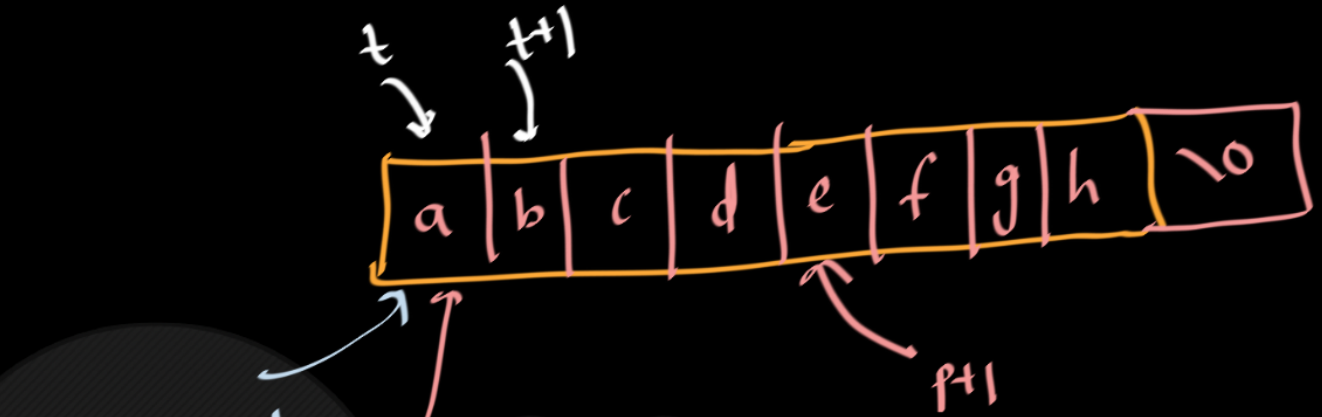
a.c

```c
#include <stdio.h>

int main()
{
    char s[] = "abcdefgh";
    char *t = s;

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


    printf("*t = %c\n", *t);
    printf("*(t+1) = %c\n", *(t+1));

}
```

Desktop — -zsh — 63×10

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c -w
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
t = 1867544056
t+1 = 1867544057
*t = a
*(t+1) = b
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

+1



$t \equiv 1000$ $\quad t+1 \equiv 1001$

$p \equiv 1000$ $\quad p+1 \equiv 1004$

GO CLASSES

a.c

```c
#include <stdio.h>

int main()
{
    char s[] = "abcdefgh";
    char *t = s;
    int *p = s; //not recommended

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

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

Desktop — -zsh — 63×10

```
(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 %
```

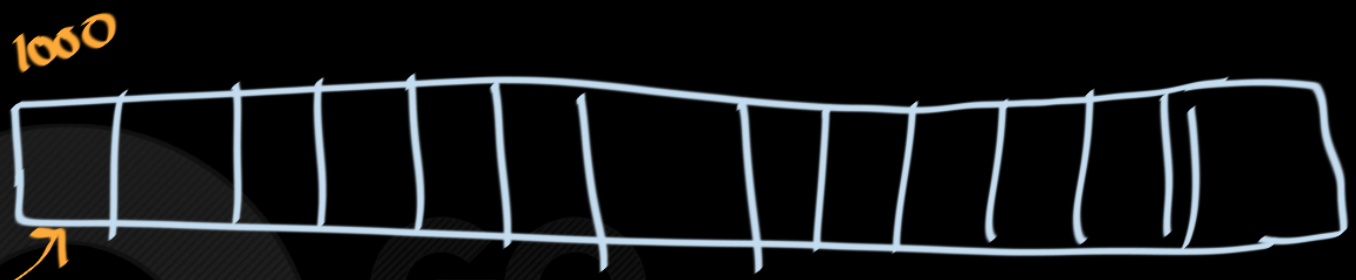
GO CLASSES

a.c

```c

int main()
{
    char s[] = "abcdefgh";
    char *t = s;
    int *p = s; //not recommended

   // t = (char *)(p+1);
    t = p+1;
    printf("*t = %c\n", *t);

}
```

Desktop — -zsh — 63×10

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.o
*t = e
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```



int *p

char *c

long int *z

c ≡ 1000

p ≡ 1000

z ≡ 1000

c+1 ≡ 1001

p+1 ≡ 1004

z+1 ≡ 1008

GO CLASSES

int *p p+1 ≡ p + sizeof(int)

char *t t+1 ≡ t + sizeof(char)

long int *z z+1 ≡ z + sizeof(long int)

Optional

q+1 ≡ q + sizeof(*q) → type that q points to

Valid arithmetic on pointers (addresses)

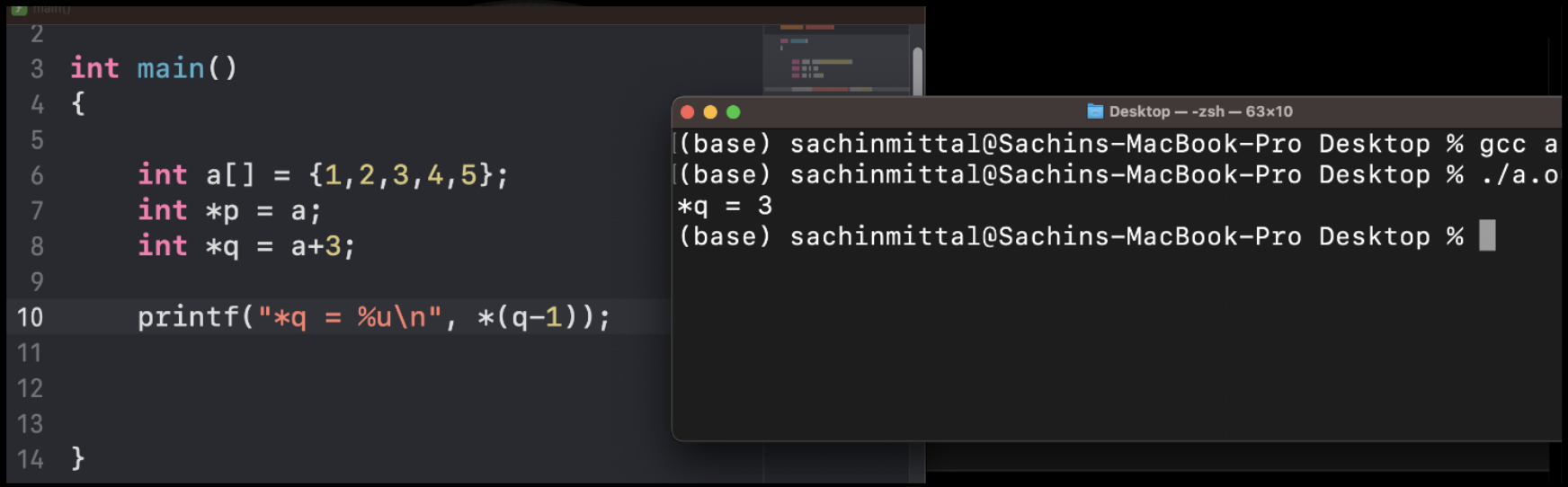
1. Adding or Subtracting an integer to the pointer
2. Difference between two pointers $P_1 - P_2$
   (Tells the distance in terms of number of elements)
3. Comparing pointers $P_1 > P_2$ $P_1 != P_2$

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

```c
int main()
{

    int a[] = {1,2,3,4,5};
    int *p = a;
    int *q = a+3;

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


}
```

```
Desktop — -zsh — 63×10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.o
*p = 1
*q = 4
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

GO CLASSES


int main()
{
int a[] = {1,2,3,4,5};
int *p = a;
int *q = a+3;
printf("*q = %u\n", *(q-1));
}
Desktop — -zsh — 63×10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.o
*q = 3
(base) sachinmittal@Sachins-MacBook-Pro Desktop %

Difference between two pointers -

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

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

$p_2 = p_1 + 3$

→ 3

$p_1 - p_2 = -3$

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.

GO CLASSES

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

```
Desktop — -zsh — 63×10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.
3
-3
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

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

$$= \frac{12}{4} = 3$$

int *R = P1+3

$P_1 \equiv 1000$

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

GO CLASSES

```c
#include <stdio.h>

int main()
{

    char s[] = "abcdef";
    char *p = s;
    char *q = s+5;

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



}
```

```
Desktop — -zsh — 63×10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.o
5
-5
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

GO CLASSES

```c
#include <stdio.h>

int main()
{

    int a[] = {1,2,3,4,5};
    int *p = a;
    int *q = a+3;

    if(p>q){
        printf("p>q \n");
    }

    if(q>p){
        printf("q>p \n");
    }
```

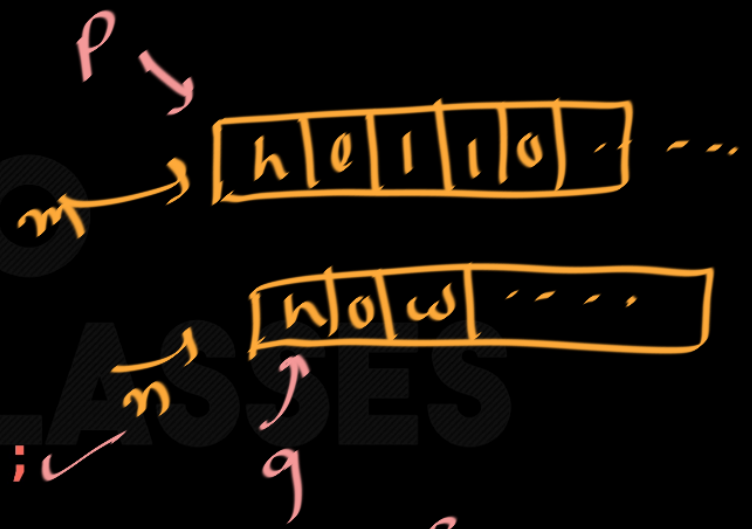
```
Desktop — -zsh — 63×10
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.o
q>p
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

GO CLASSES

```c
#include <stdio.h>

int main()
{

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


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



}
```

```
Desktop — -zsh — 72×14
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.
1
2
3
4
5
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

```c
#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 are differ

Invalid arithmetic on pointers

```
p1 + p2
p1 * p2
p1 % p2
p1 / p2
```

adding 2 addresses doesn't make sense

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

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

```c
#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;
}
```

https://cs.uwaterloo.ca/~cbruni/CS137Resources/lectures/CS137Part05_pointers_arrays_malloc_post.pdf

Question

```c
#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;
}
```

https://cs.uwaterloo.ca/~cbruni/CS137Resources/lectures/CS137Part05_pointers_arrays_malloc_post.pdf