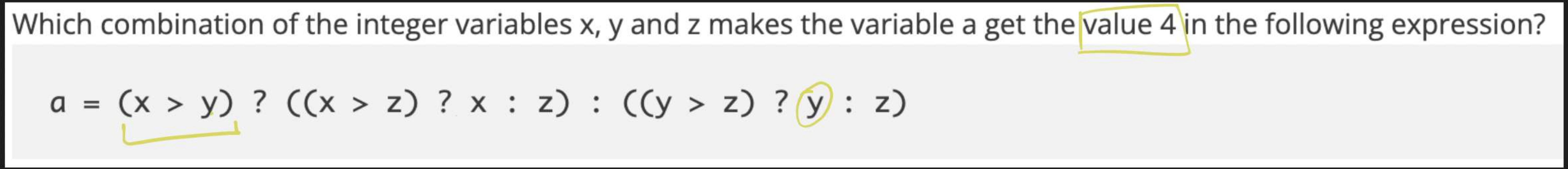
Practice Session - III (GATE PYQs)

Saturday, 29 June 2024 7:08 PM

1.

GATE CS, 08



Which combination of the integer variables x, y and z makes the variable a get the value 4 in the following expression?

```
a = (x > y) ? ((x > z) ? x : z) : ((y > z) ? y : z)
```

[✓ a)] $x = 3, \; y = 4, \; z = 2$

b) $x = 6, \; y = 5, \; z = 3$

c) $x = 6, \; y = 3, \; z = 5$

d) $x = 5, \; y = 4, \; z = 5$

Sol.

$a \rightarrow \boxed{a = 4}$

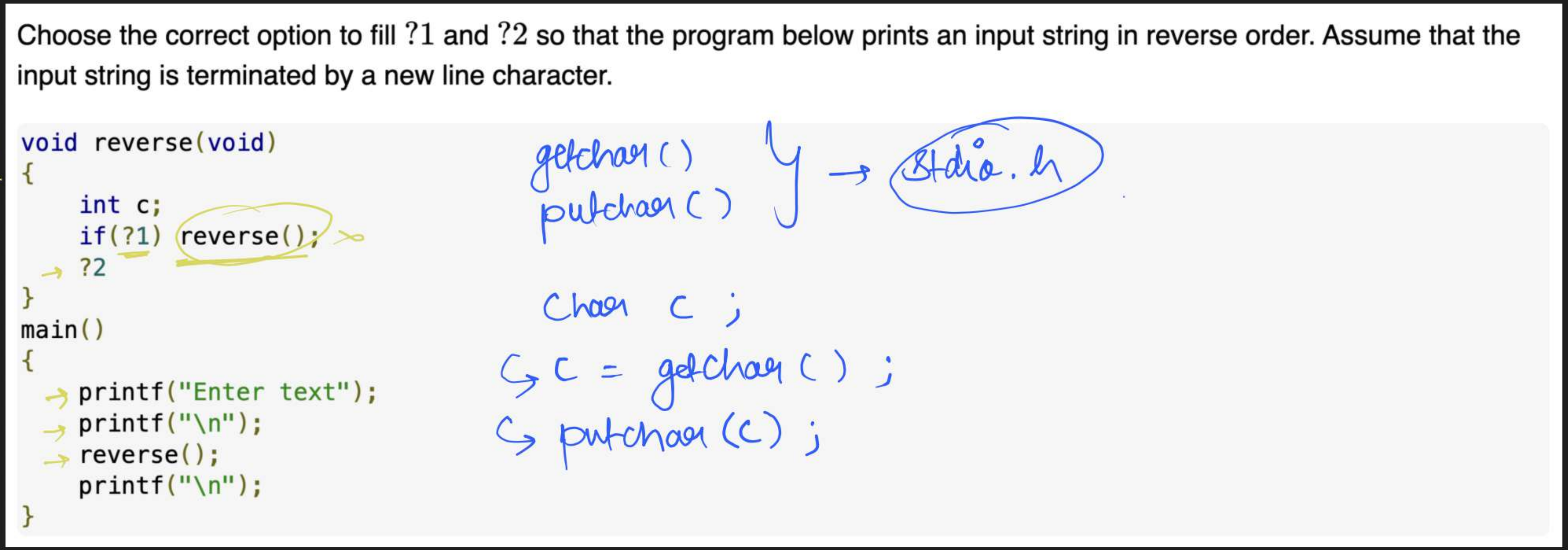
$b \rightarrow a = 6$

$c \rightarrow a = 6$

$d \rightarrow a = 5$

2.

GATE CS, 08



Choose the correct option to fill ?1 and ?2 so that the program below prints an input string in reverse order. Assume that the input string is terminated by a new line character.

```
void reverse(void)
{
    int c;
    if(?1) reverse();
    ?2
}
main()
{
    printf("Enter text");
    printf("\n");
    reverse();
    printf("\n");
}
```

getchar( )
putchar( ) } → stdio.h

char c ;

↳ c = getchar ( ) ;

↳ putchar (c) ;

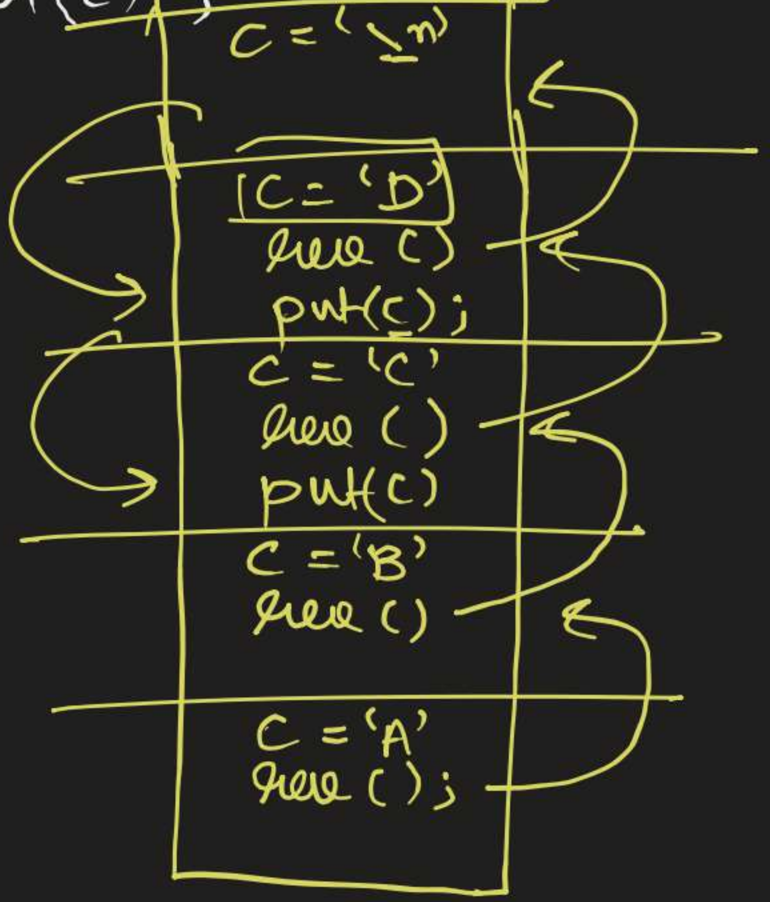
a) 1 → getchar ( ) ! = '\n'
2 → getchar (c) ;

b) 1 → (c = getchar ( )) ! = '\n'
2 → getchar (c) ;

c) 1 → c ! = '\n'
2 → putchar (c) ; }

d) 1 → (c = getchar ( )) ! = '\n'
2 → putchar(c) ;

→ A, B, C, D, \n

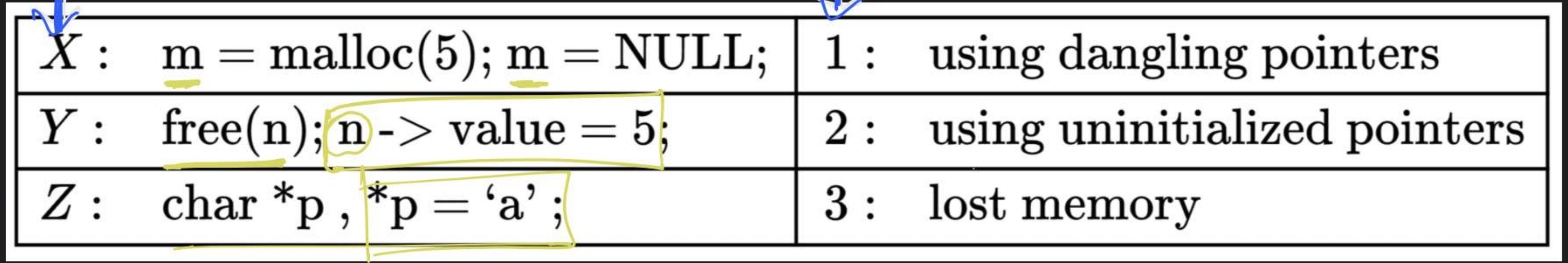


'\n'

D, C, B, A

③

GATE CS, 2000



X :	m = malloc(5); m = NULL;	1 :	using dangling pointers
Y :	free(n); n -> value = 5;	2 :	using uninitialized pointers
Z :	char *p , *p = ‘a’ ;	3 :	lost memory

a) X-1, Y-3, Z-2

b) X-2, Y-1, Z-3

c) X-3, Y-2, Z-1

[✓ d)] X-3, Y-1, Z-2

Sol:

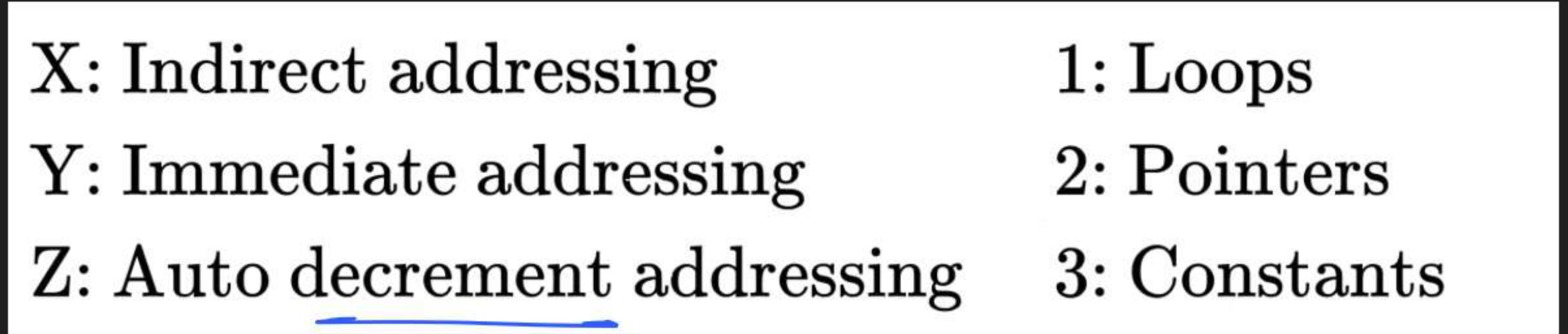
X : memory is allocated to m, but not freed.
So this chunk of memory can't be allocated to some other variable.
m = null, there is no way to access that allocated memory.

Y : memory freed, but still the pointer exists. → dangling pointer.

Z : p remains uninitialized, also called wild pointers.

4.

GATE CS, 2000.



X: Indirect addressing	1: Loops
Y: Immediate addressing	2: Pointers
Z: Auto decrement addressing	3: Constants

a) X - 3, Y - 2, Z - 1

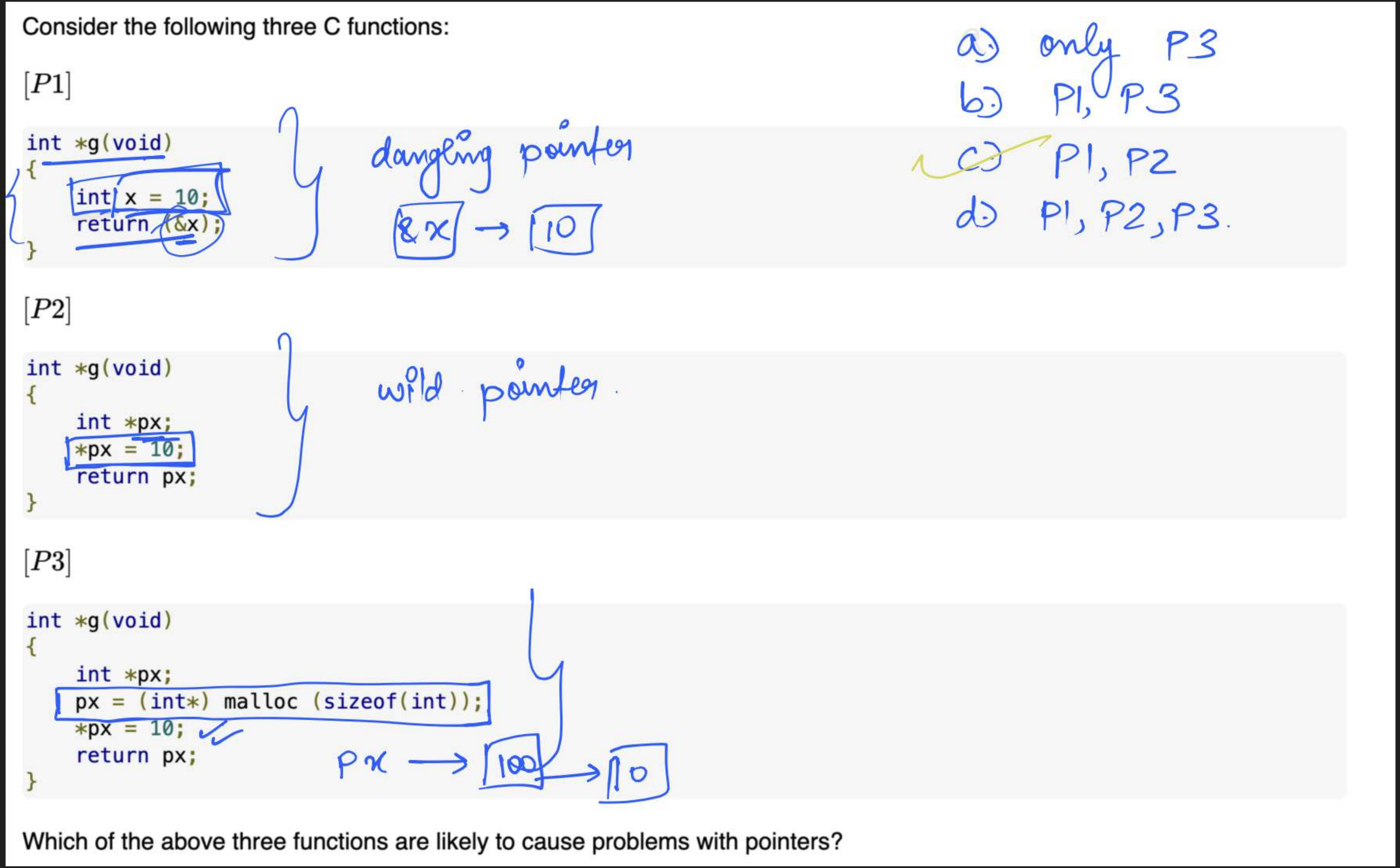
b) X - 1, Y - 3, Z - 2

[✓ c) X - 2, Y - 3, Z - 1]

d) X - 3, Y - 1, Z - 2

5.

GATE CS, 01



Consider the following three C functions:

[P1]

```
int *g(void)
{
    int x = 10;
    return (&x);
}
```

dangling pointer

&x → 10

[P2]

```
int *g(void)
{
    int *px;
    *px = 10;
    return px;
}
```

wild pointer.

[P3]

```
int *g(void)
{
    int *px;
    px = (int*) malloc (sizeof(int));
    *px = 10;
    return px;
}
```

px → 100 → 10

Which of the above three functions are likely to cause problems with pointers?

a) only P3

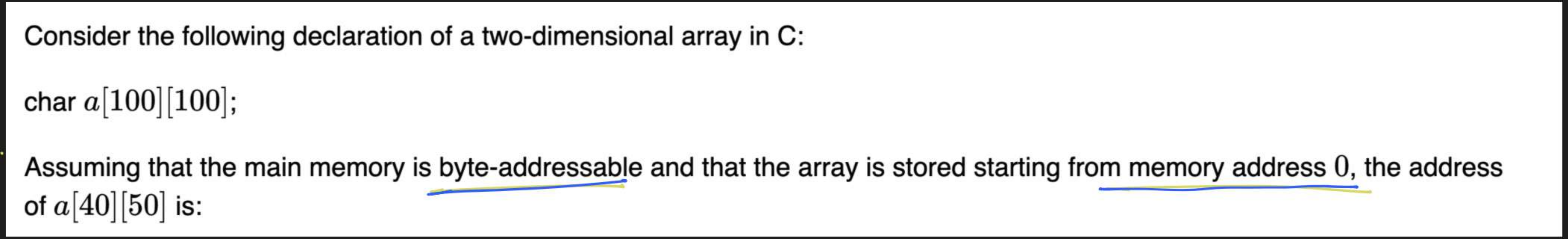
b) P1, P3

[✓ c) P1, P2]

d) P1, P2, P3.

⑥

GATE
CS,02.



Consider the following declaration of a two-dimensional array in C:

char $a[100][100]$;

Assuming that the main memory is byte-addressable and that the array is stored starting from memory address 0, the address of $a[40][50]$ is:

a) 4040 c) 5040

b) ✓ 4050 d) 5050

Sol.

C → follows row-major form for storing arrays.

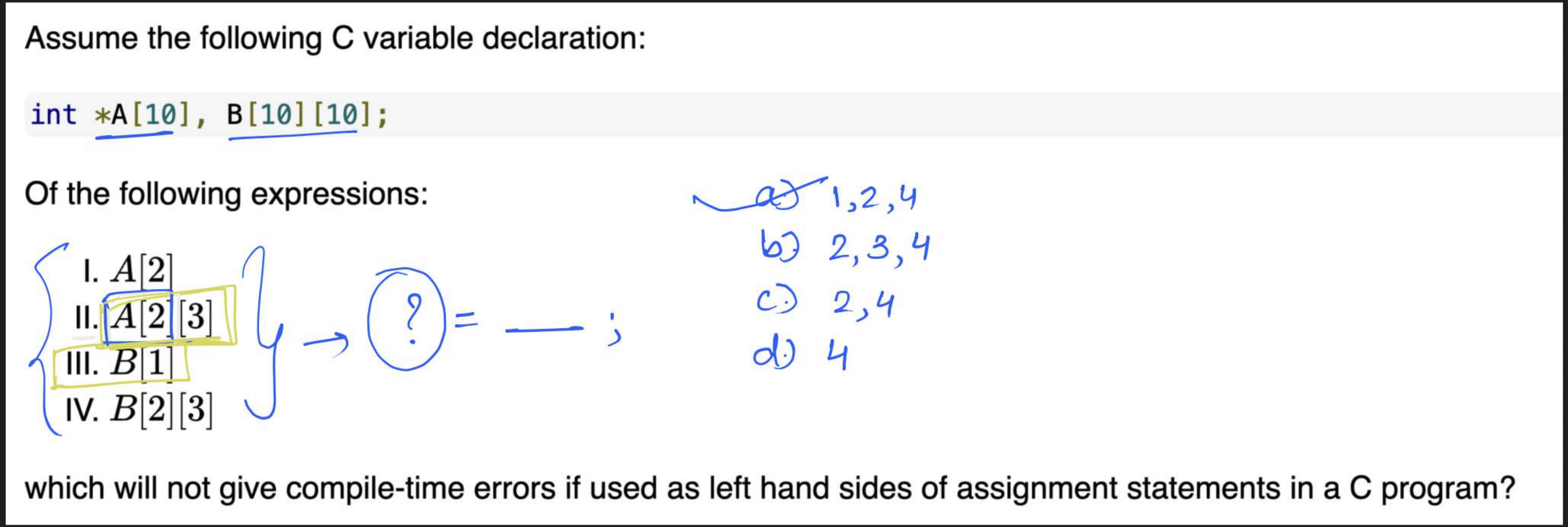
$$loc(a[i][j]) = BA + [i * C + j] * m \quad ; \quad \begin{aligned} m &= 1 \\ BA &= 0 \\ i &= 40, \ j = 50 \\ C &= 100 \end{aligned}$$

$$= 0 + [40 * 100 + 50] * 1$$

$$= \boxed{4050}$$

7

GATE CS, 03



Assume the following C variable declaration:

```
int *A[10], B[10][10];
```

Of the following expressions:

I. $A[2]$
II. $A[2][3]$
III. $B[1]$
IV. $B[2][3]$

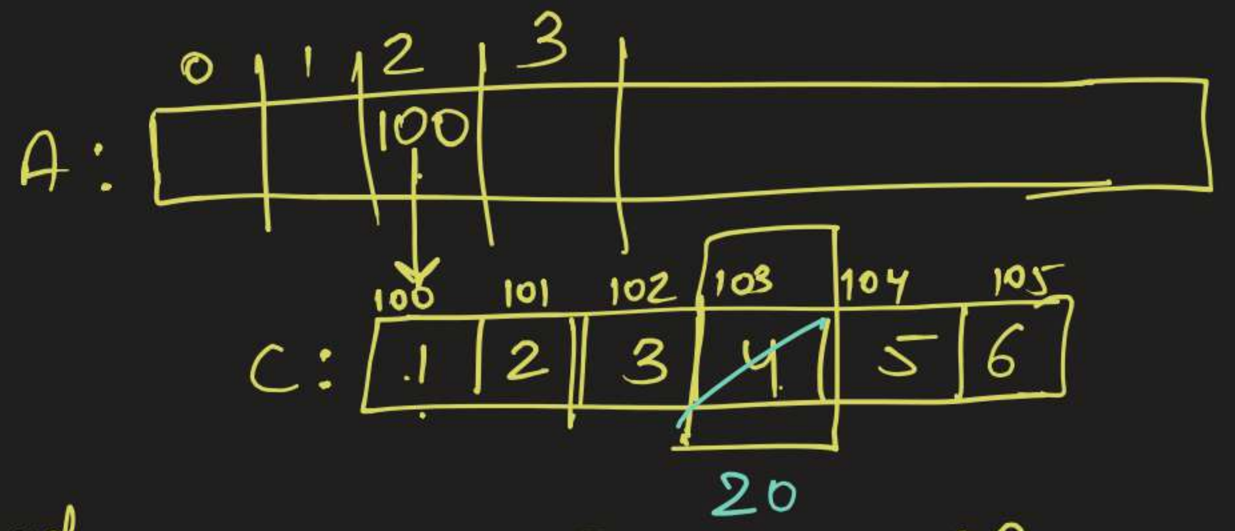
→ ? = ___ ;

a) 1,2,4
b) 2,3,4
c) 2,4
d) 4

which will not give compile-time errors if used as left hand sides of assignment statements in a C program?

Sol.

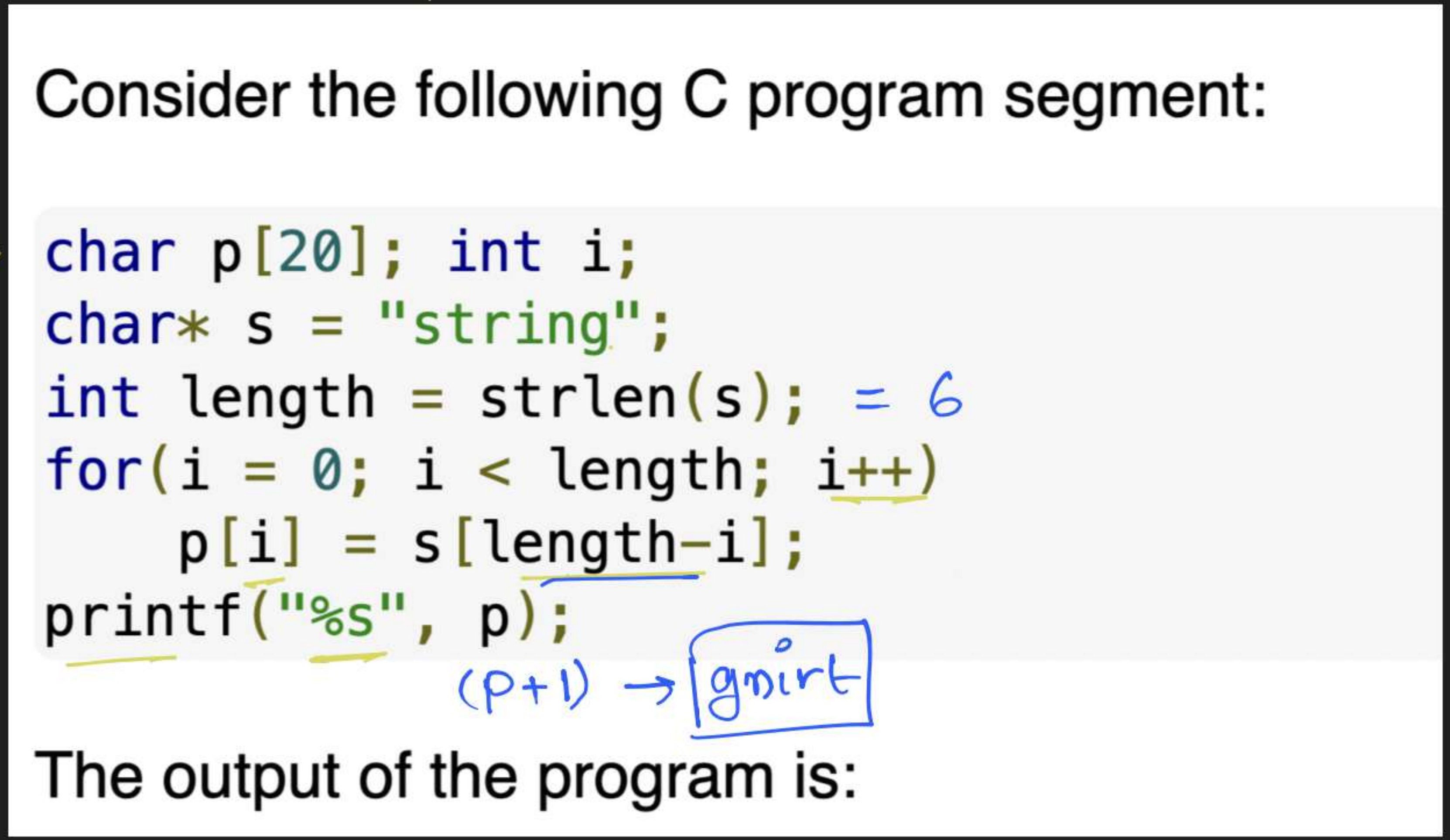
```
int C[] = {1, 2, 3, 4, 5, 6};
A[2] = C;
A[2][3] = 20;
```



B[1] → is the address of the 2nd row in B. [10x10 matrix].
& we can't directly change the address of any row in a 2-D array.

8.

GATE CS, 04.



Consider the following C program segment:

```
char p[20]; int i;
char* s = "string";
int length = strlen(s); = 6
for(i = 0; i < length; i++)
    p[i] = s[length-i];
printf("%s", p);
```

(p+1) → gnirt

The output of the program is:

a) gnirts

b) String

c) gnirt

[✓ d) no output]

\0

0	1	2	3	4	5	6
s	t	r	i	n	g	\0

s →

0	1	2	3	4	5	6	7
\0	g	n	i	r	t		

p →

Sol.

$i = 0,\quad P[0] = S[6-0] = S[6]$

$P[0] = '\backslash 0';$

$i = 1;\quad P[1] = S[5] = 'g';$

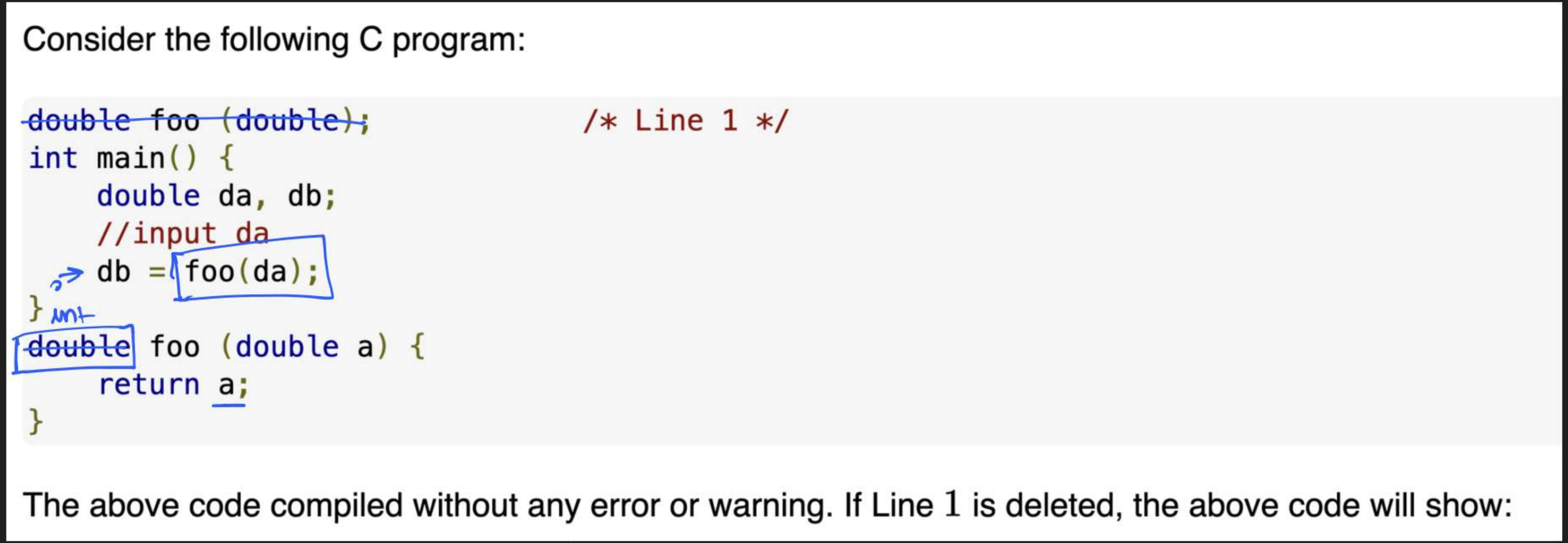
$\vdots$

$i = 5;\quad P[5] = S[1] = 't';$

$i = 6;$ for terminates.

9.

GATE CS, 05



Consider the following C program:

```
~~double foo (double);~~                    /* Line 1 */
int main() {
    double da, db;
    //input da
  ?→ db = foo(da);
} int
~~double~~ foo (double a) {
    return a;
}
```

The above code compiled without any error or warning. If Line 1 is deleted, the above code will show:

a) no compilation error or warning.

b) Some compiler warnings not leading to unintented results.

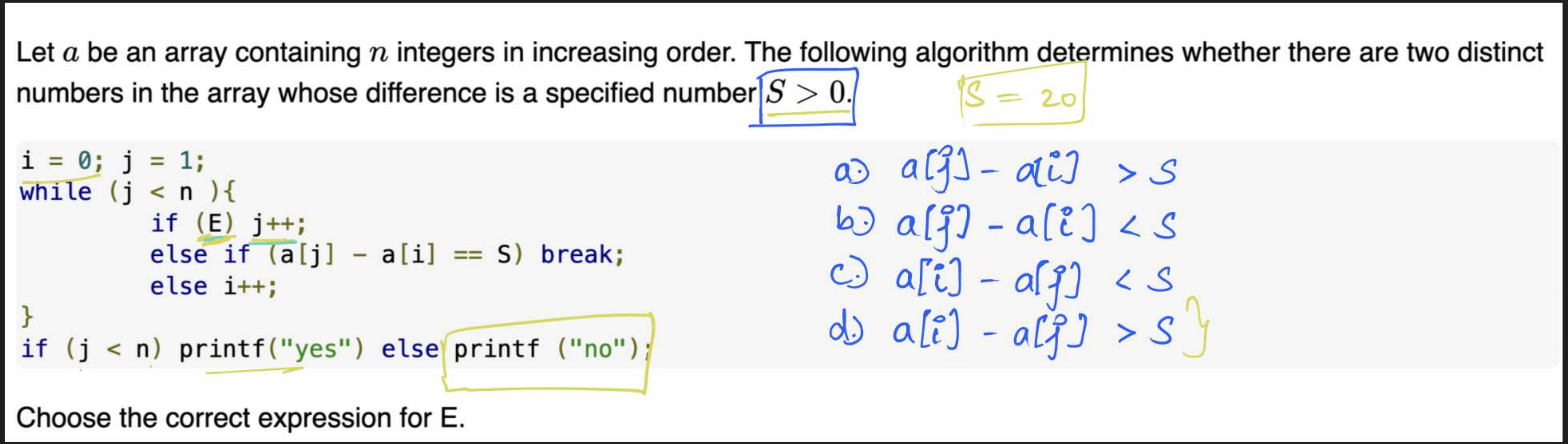
c) " " " due to type- mismatch eventually leading to unintented results.

[✓ d) Compilation error.]

Sol. compilation error → conflicting types (type mismatch).

(10)

GATE CS, 05



Let a be an array containing n integers in increasing order. The following algorithm determines whether there are two distinct numbers in the array whose difference is a specified number $\boxed{S > 0}$.

```
i = 0; j = 1;
while (j < n ){
          if (E) j++;
          else if (a[j] - a[i] == S) break;
          else i++;
}
if (j < n) printf("yes") else printf ("no");
```

Choose the correct expression for E.

$S = 20$

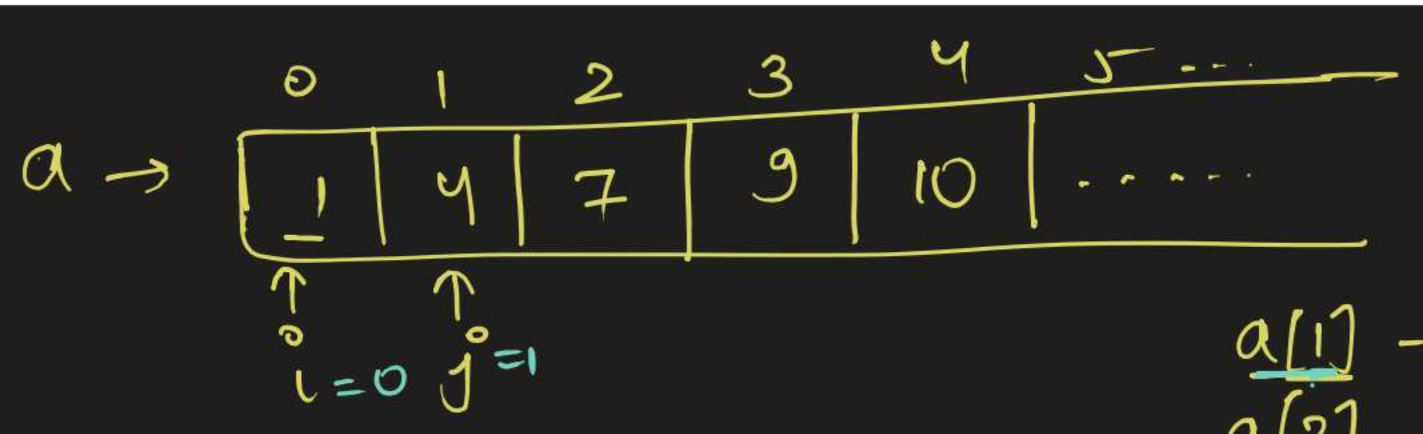
a) $a[j] - a[i] > S$

b) $a[j] - a[i] < S$

c) $a[i] - a[j] < S$

d) $a[i] - a[j] > S$

Sol.



$a[1] - a[0] = x$

$a[2] - a[0] = y > x$

a) $a[j] - a[i] > S$ → this will always be true.

b) $a[j] - a[i] \boxed{< S}$

$a(1) - a(0) = x < S$

$a(2) - a(0) = y > x < S = S > S$

1 - 4

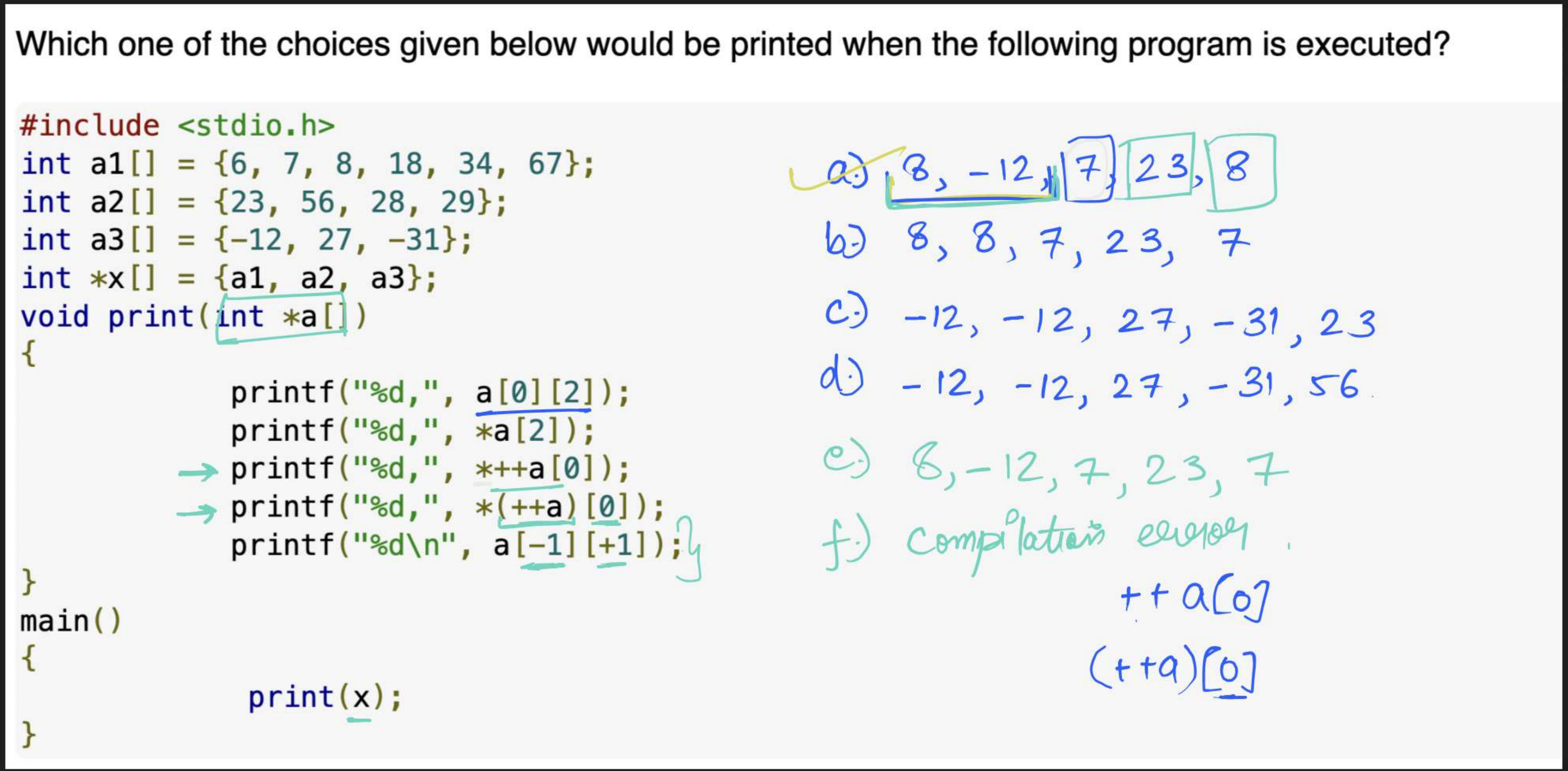
c) $\boxed{\underbrace{a[i] - a[j]}_{-ve.} < S \; ; \quad \& \; S > 0}$

d) $a[i] - a[j] > S$ → it is incorrect because some comparisons of a[i] & a[j] are left.

$\boxed{\{a[2] - a[0]\}}$

11.

GATE CS, 06



Which one of the choices given below would be printed when the following program is executed?

```
#include <stdio.h>
int a1[] = {6, 7, 8, 18, 34, 67};
int a2[] = {23, 56, 28, 29};
int a3[] = {-12, 27, -31};
int *x[] = {a1, a2, a3};
void print(int *a[])
{
            printf("%d,", a[0][2]);
            printf("%d,", *a[2]);
            printf("%d,", *++a[0]);
            printf("%d,", *(++a)[0]);
            printf("%d\n", a[-1][+1]);
}
main()
{
             print(x);
}
```

a) 8, -12, 7, 23, 8

b) 8, 8, 7, 23, 7

c) -12, -12, 27, -31, 23

d) -12, -12, 27, -31, 56

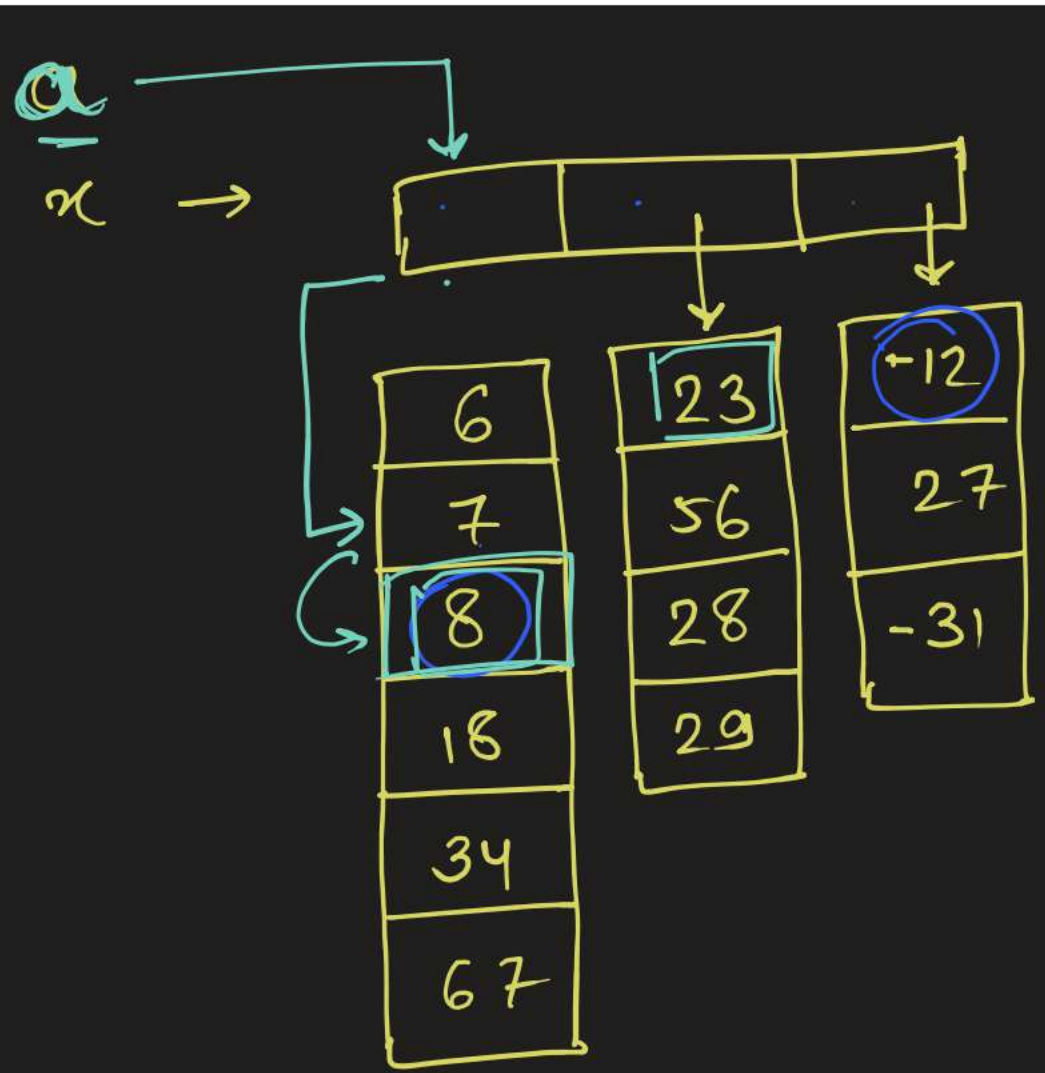
e) 8, -12, 7, 23, 7

f) compilation error

++a[0]

(++a)[0]

Sol =



$(*) < (++)$

$x[0] \rightarrow a_1[0]$

$\hookrightarrow a_1[1]$