Operators in C programming

Operator

- Arithmetic Operators
- Relational Operators
- Logical Operators

.....

Arithmetic Operators

Operator	Name	Description
+	Addition	to add two numbers together
-	Subtraction	to subtract one number from another
*	Multiplication	to multiply one number by another.
/	Division	to divide one number by another.
%	Modulus (Remainder)	to find the remainder from dividing one number by another.

* / %	Multiplication, division and modulus	left to right
+ -	Addition and subtraction	left to right

3+2*5

2*3/6 = 1

2*(3/6) = 0 X

Precedence of Arithmetic Operators:

- High priority $*$,/, %
- Low priority $+, -$

- The basic evaluation procedure includes two left-to-right passes through the expression.

- During the first pass, the high priority operators are applied as they are encountered.

- During the second pass, the low priority operators are applied as they are encountered.

```
int a = 9 - 12/3 + 3 * 2 - 1
```

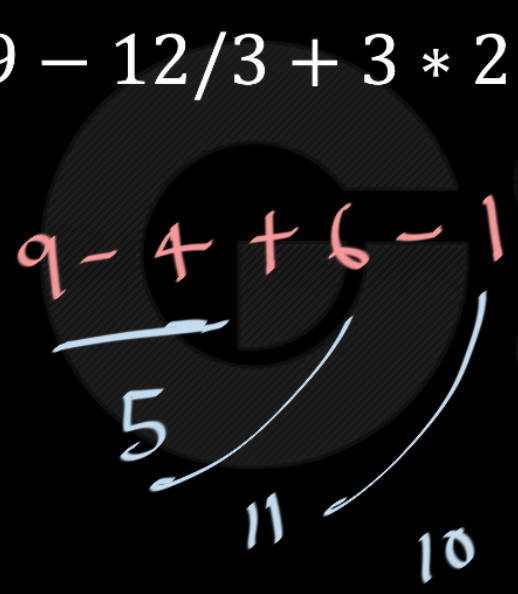
$= ((9 - 4) + 6) - 1$

$(5 + 6) - 1$

$11 - 1 \quad = 10$

```
int a = 9 – 12/3 + 3 * 2 – 1
```

```
int a = 9 – 12/3 + 3 * 2 – 1
```



First pass

Step1: x = 9–4+3*2–1

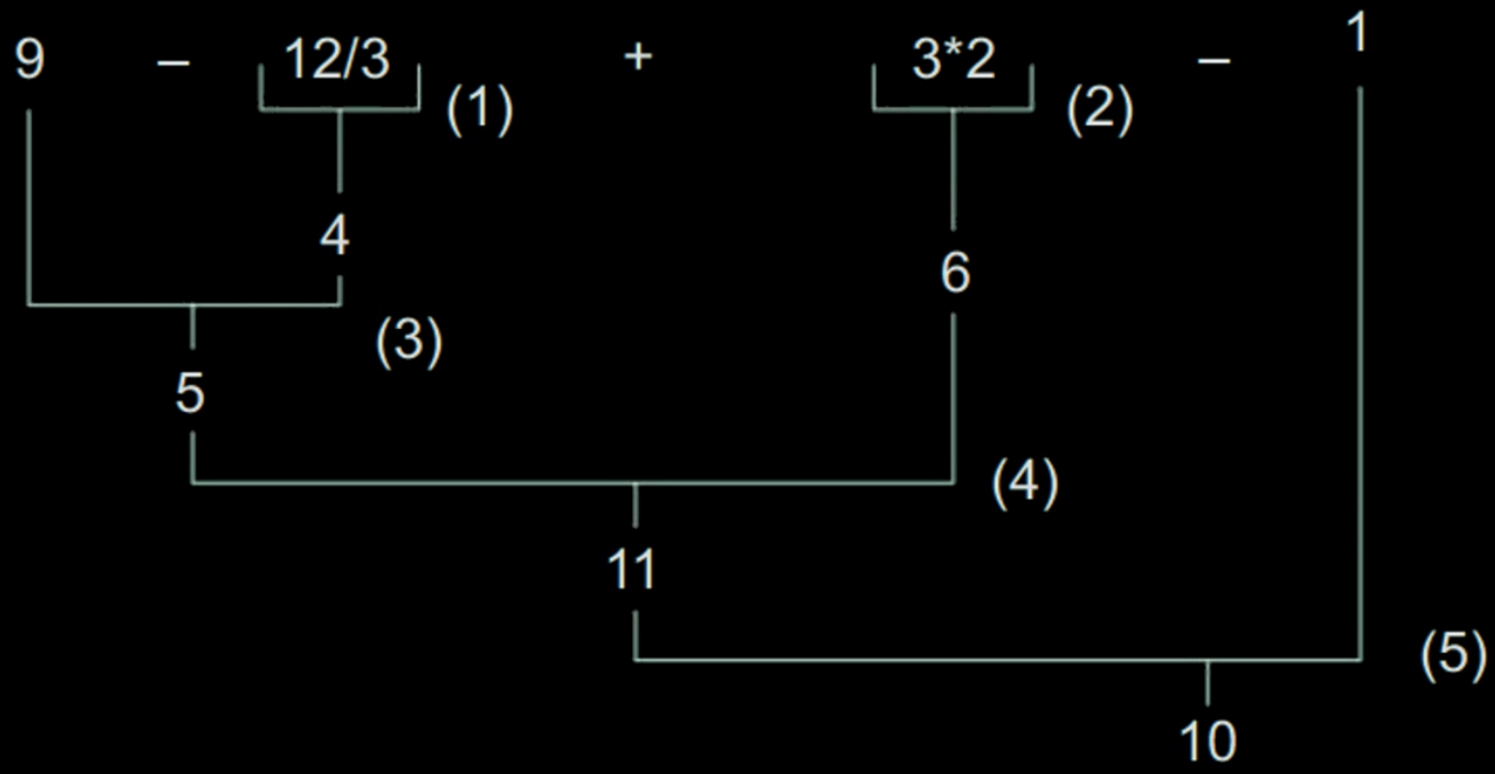
Step2: x = 9–4+6–1

Second pass

Step3: x = 5+6–1

Step4: x = 11–1

Step5: x = 10


9
–
12/3
(1)
+
3*2
(2)
–
1
4
6
5
(3)
(4)
11
(5)
10

Relational Operators

Operator	*Meaning*
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

< <= > >=	relational less than/less than equal to relational greater than/greater than or equal to	left to right
== !=	Relational equal to or not equal to	left to right

Question 1

```c
#include <stdio.h>


int main() {
    int a = 10, b = 20, c = 30;
    if (a < b < c) {
        printf("True\n");
    } else {
        printf("False\n");
    }
    return 0;
}
```

10 < 20 < 30

1 < 30

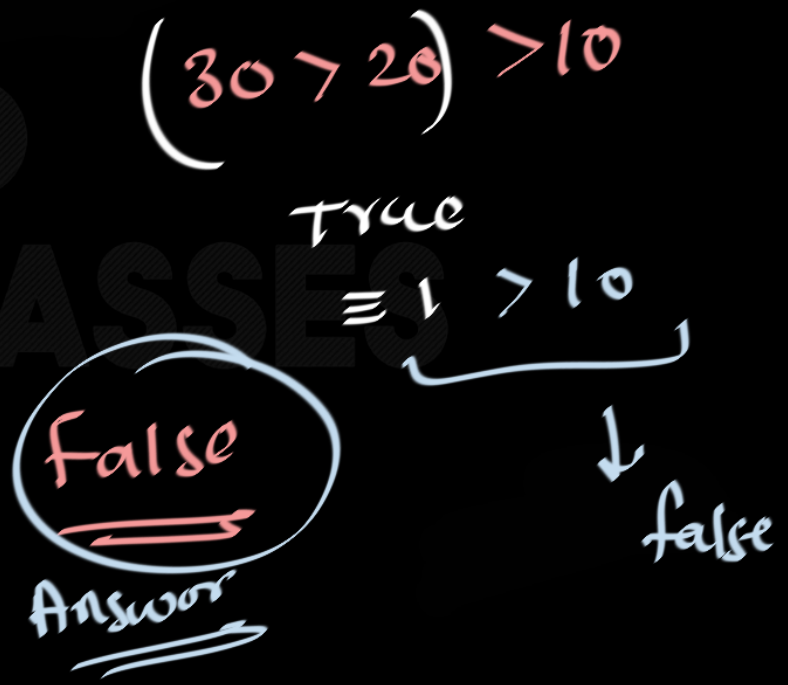
true

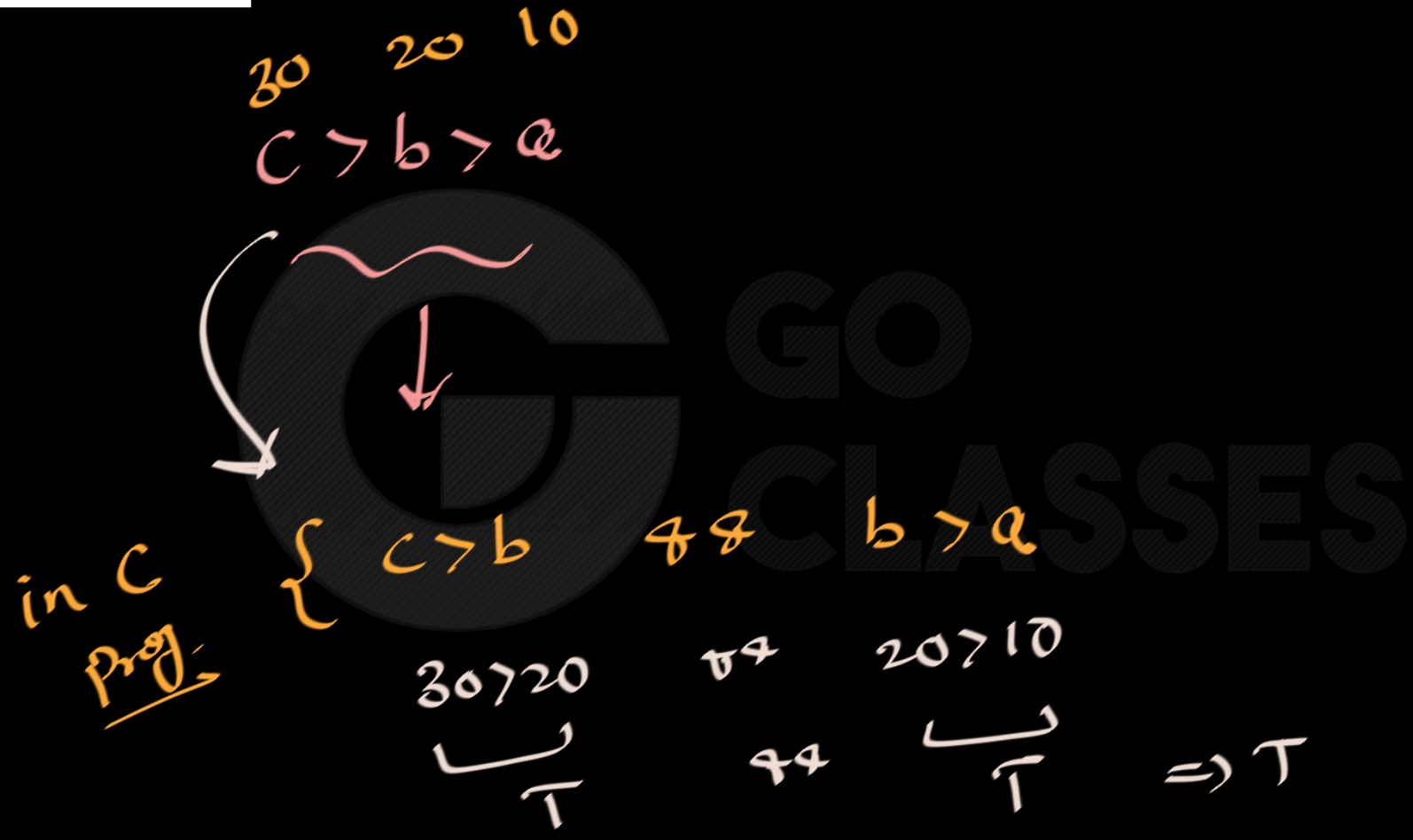
True

Good program practice → { a<b && b<c

Question 2

```c
#include <stdio.h>

int main() {
    int a = 10, b = 20, c = 30;
    if (c > b > a) {
        printf("True\n");
    } else {
        printf("False\n");
    }
    return 0;
}
```




30 20 10
c > b > a
in C
Prog.
{ c > b && b > a
30 > 20 && 20 > 10
T && T => T

LOGICAL OPERATORS

&&	meaning logical	AND
||	meaning logical	OR
!	meaning logical	NOT

!	not operator	right to left
	... ...	
&&	Logical AND	left to right
||	Logical OR	left to right

```c
#include<stdio.h>
int main() {
    int x = 1, y=2;

    if (x < 10 && y > 0)
        printf("Welcome");
    else
        printf("Awesome");


    return 0;
}
```

CLASSES

```
int x = 10, y = 20, z = 1;

if (x < 1 && y > 50 || z < 10)
    printf("True");
else
    printf("False");
```

&&	Logical AND	left to right
||	Logical OR	left to right

```
int x = 10, y = 20, z = 1;

if (x < 1 && y > 50 || z < 10)
     printf("True");
else
     printf("False");
```

((x < 1 && y > 50) || z < 10)

((F && F) || T)

F || T => T

GO CLASSES

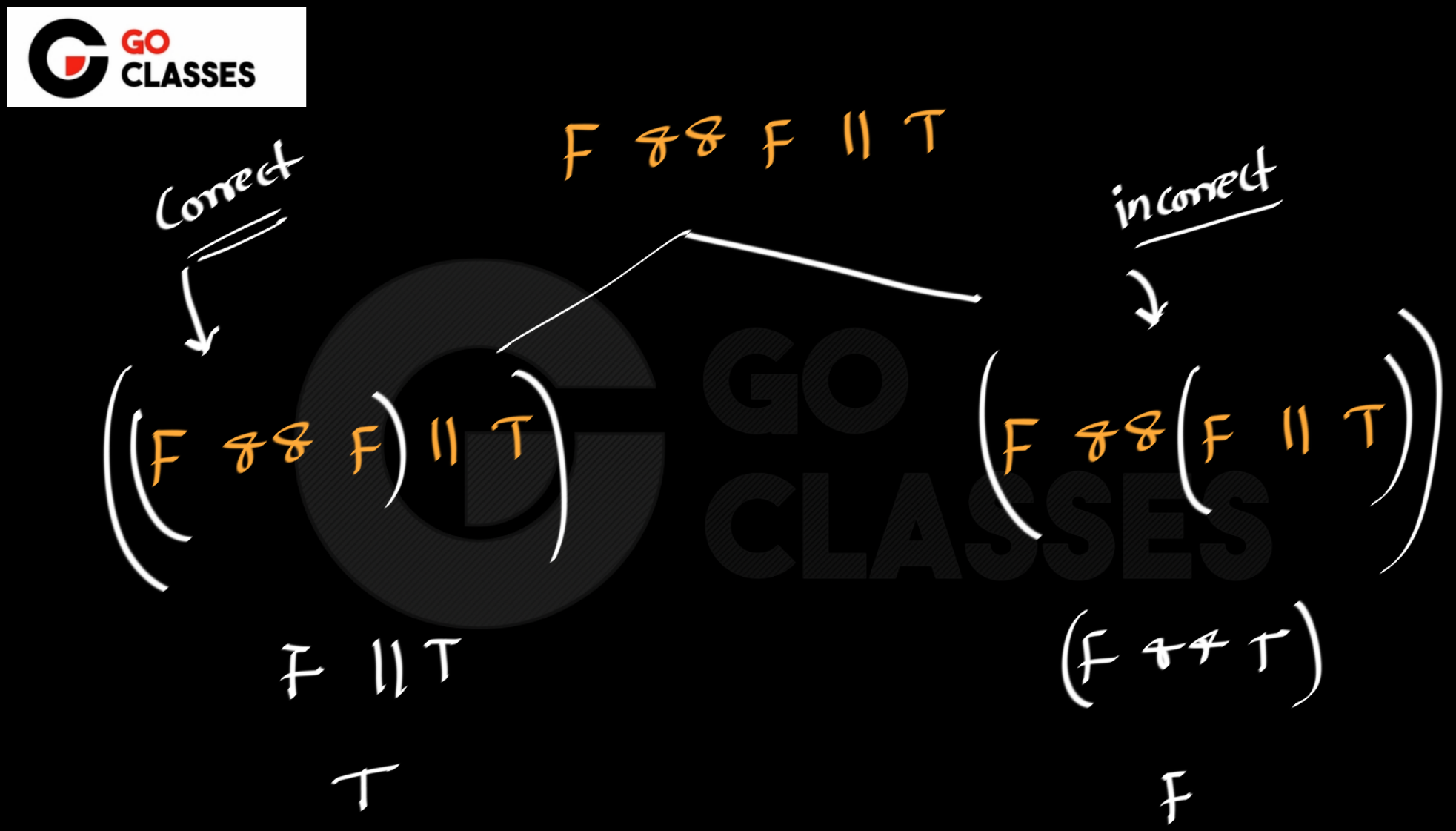
By mistake

```
x < 1 && (y > 50 || z < 10)
```

F F T

F && (F || T)

F && T ⇒ F


F && F || T
Correct
incorrect
((F && F) || T)
(F && (F || T))
F || T
T
(F && T)
F

GO CLASSES

Q: exp1 || exp2 || exp3

Put the brackets correctly

Answer: ((exp1 || exp2) || exp3)

Q: exp1 && exp2 && exp3

Put the brackets correctly

((exp1 && exp2) && exp3)

GO CLASSES

Q: exp1 && exp2 || exp3

Put the brackets correctly

Answer: ((exp1 && exp2) || exp3)

GO CLASSES

Q: exp1 || exp2 && exp3

Put the brackets correctly

Answer: (exp1 || (exp2 && exp3))

GO CLASSES

Q: exp1 && exp2 || exp3 && exp4

Put the brackets correctly

Answer: ((exp1 && exp2) || (exp3 && exp4))

GO CLASSES

Q: expr1 || expr2 && expr3 || expr4

Put the brackets correctly

Soln ((expr1 || (expr2 && expr3)) || expr4)

GO CLASSES

order of evaluation

exp1 || (exp2 && exp3)

Increment and Decrement operators

```
++m; or m++;
--m; or m--;
```

++m; is equivalent to m = m+1; (or m += 1;)

--m; is equivalent to m = m-1; (or m -= 1;)

```
n = 5;

x = n++;

x = ++n;
```

```
x = n;
n = n+1;
```

```
int main()
{
    int n = 5;
    int x;
    x = n++;
    pf(x)
}
```

GO CLASSES

```
int main ( )
{
    int n = 5;
    int x;
    x = n++;
    Pf (x) ⟶ 5
    Pf (n) ⟶ 6
}
```

Post - increment

GO CLASSES

```
int main( )
{
    int n = 5;
    int x;
    x = ++n;
    Pf(x) ⟶ 6
    Pf(n) ⟶ 6
}
```

Pre-increment

++2

++(a+b*2)

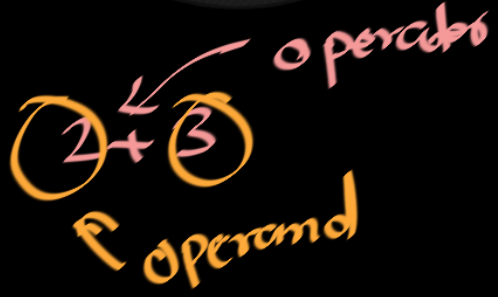
expression

Bitwise Operators

- Bitwise AND (&)
- Bitwise OR (|)
- Bitwise exclusive OR (^)
- Left Shift <<
- Right Shift >>
- One's complement ~

x y XOR

op1	op2	op1 & op2	op1 | op2	op1 ^ op2
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0



x op y

Bitwise And : &

$2^3\ 2^2\ 2^1\ 2^0$

1 1 0 1

x - - -> 0000 0000 0000 1101 → 13

y - - -> 0000 0000 0001 1001 → 25

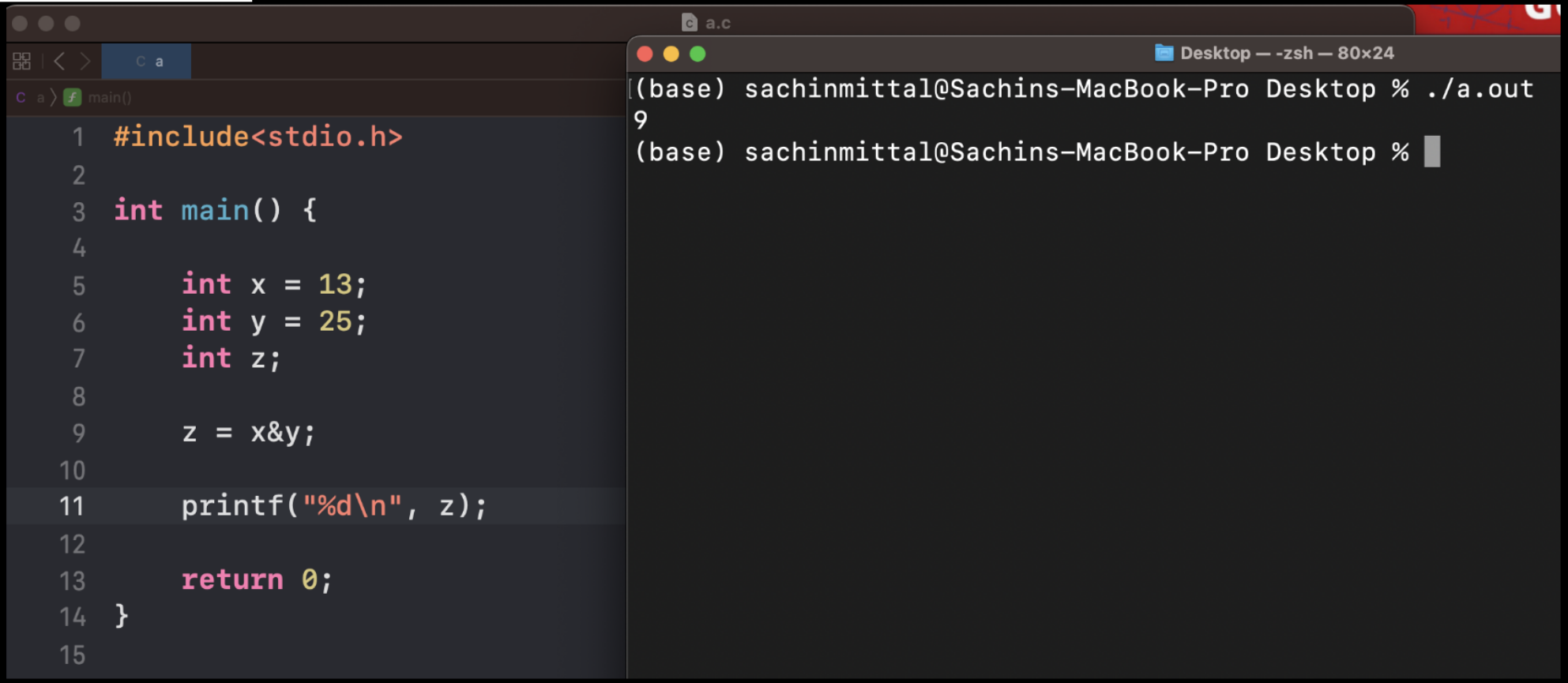
2^4

z = x & y ;

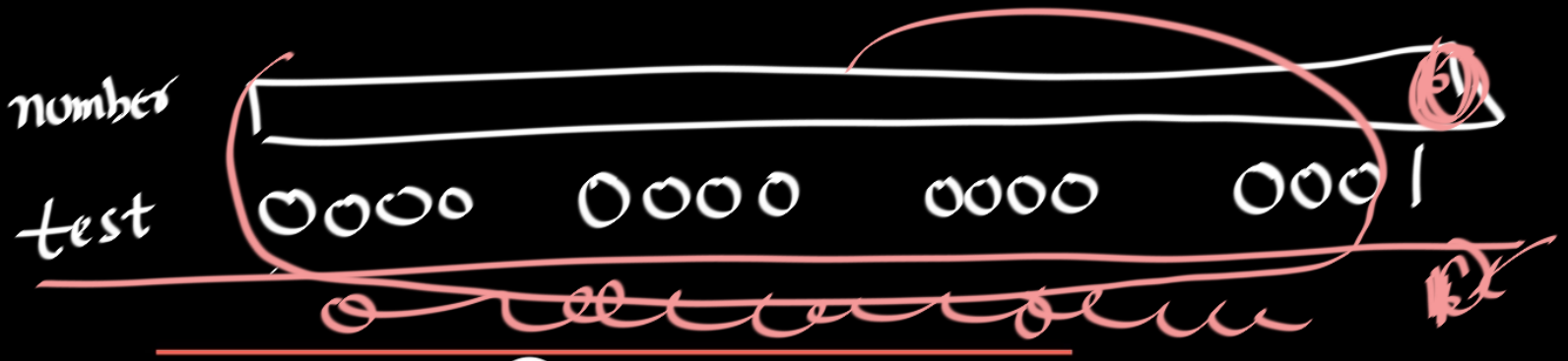
16
9
25

z - - -> 0000 0000 0000 1001

↳ 9


a.c
c a
c a > main()
1 #include<stdio.h>
2
3 int main() {
4
5 int x = 13;
6 int y = 25;
7 int z;
8
9 z = x&y;
10
11 printf("%d\n", z);
12
13 return 0;
14 }
15
Desktop — -zsh — 80×24
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
9
(base) sachinmittal@Sachins-MacBook-Pro Desktop %

```
int test = 1;

if(number & test)
    print("Number is odd\n\n");
else
    printf("Number is even\n\n");
```



Bitwise OR : |

x - - -> 0000 0000 0000 1101 → 13

y - - -> 0000 0000 0001 1001 → 25

---

x|y - - -> 0000 0000 0001 1101 → 29

---

16 8 4 1

Bitwise Exclusive OR: ^

x - - ->	0000 0000 0000 1101
y - - ->	0000 0000 0001 1001
x^y - - ->	0000 0000 0001 0100

Left shift and Right shift

Left shift: $op << n$

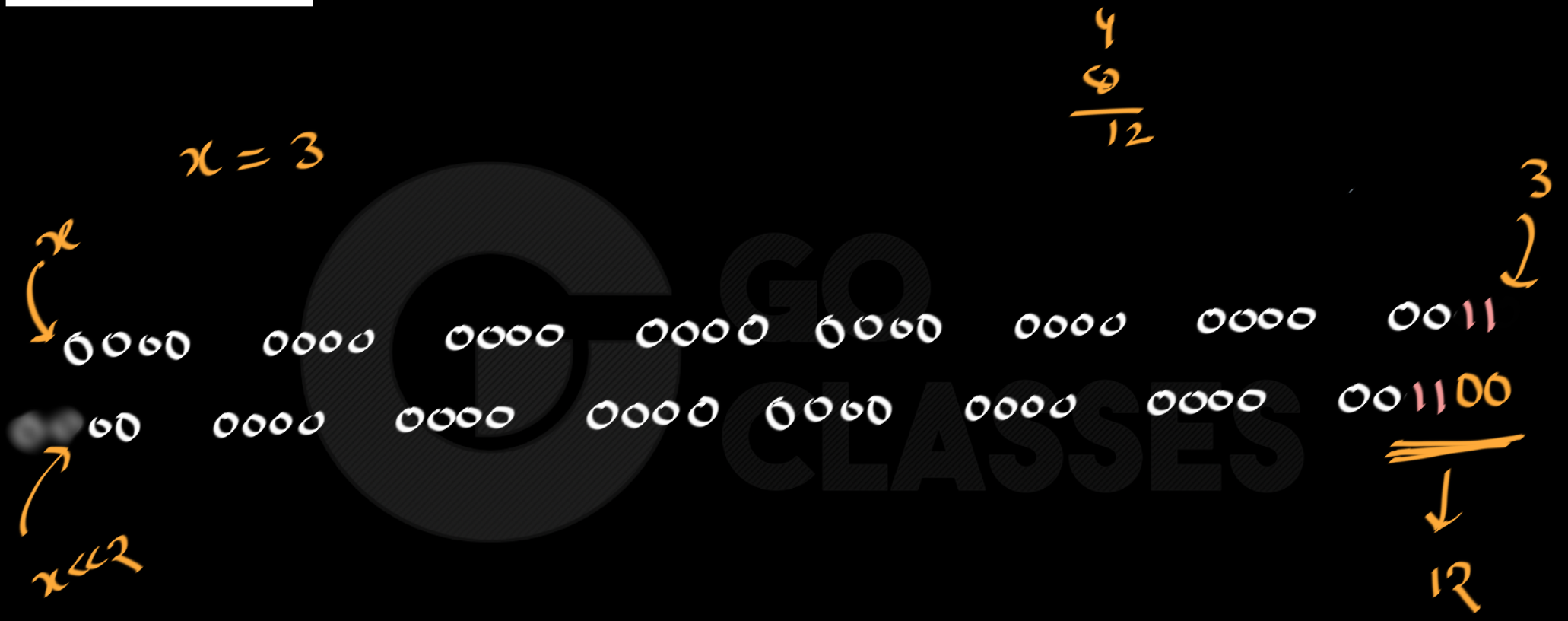
Right shift: $op >> n$

Left shift

suppose x is 0100 1001 1100 1011

x << 3 = 0100 1110 0101 1000


x = 0100 1001 1100 1011
x << 3 = 0100 1001 1100 1011000

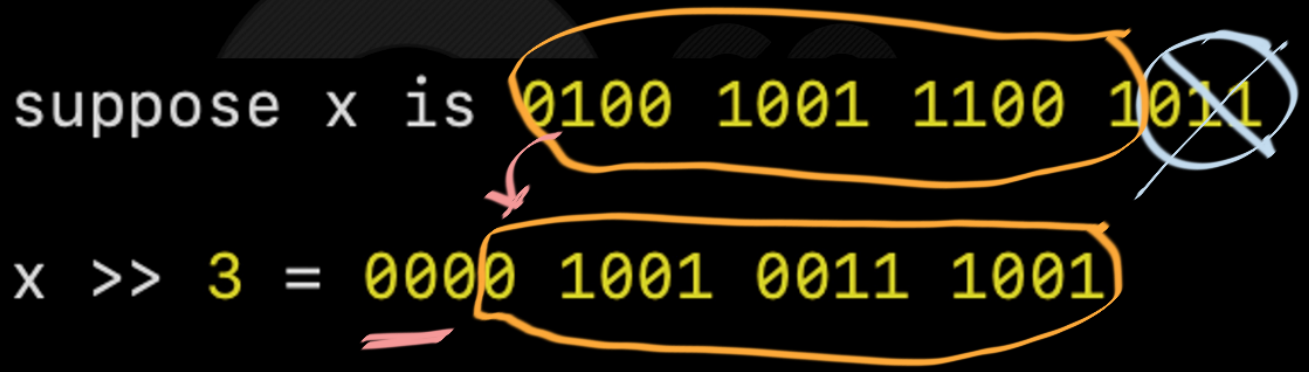

x = 3
4
3
12
x
3
0000 0000 0000 0000 0000 0000 0000 0000 0011
00 0000 0000 0000 0000 0000 0000 001100
x<<2
12

Right Shift

suppose x is 0100 1001 1100 1011

x >> 3 = 0000 1001 0011 1001

Right Shift



Right Shift

```
suppose x is 0100 1001 1100 1011

x >> 3 = 0000 1001 0011 1001
```

Unsigned Right shift: - Fill zeros

signed Right shift:- Depends on system - either zeros or sign bit

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

    int x,y =10;
    x = y << 1;
    printf("%d",x);
}
```

↳ 20

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

    int x,y =10;
    x = y >> 1;
    printf("%d",x);
}
```

↳ 5

One's complement operator: ~

```
x = 1001 0110 1100 1011

~x = 0110 1001 0011 0100
```

! True = false

! 10 = 0

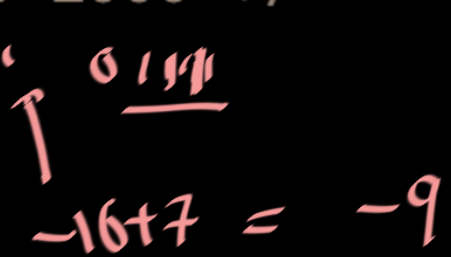
! 0 = 1

One's complement work on the bit level.

x =

Example -

```
int main(){

    int x;

    x = 8; /* 0000 0000 0000 1000 */

    printf("%d",  ~x);
}
```



$-x = \sim x + 1$ (in 2's Complemented System)

$\sim x + 1 - \sim x = 1$

$\sim x + 1$: 2's comp

$-x - \sim x = 1$

GO CLASSES

```c
#include<stdio.h>

int main() {

    int x = 8;


    printf("%d\n",-x-~x);


    return 0;
}
```

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

$\sim x + 1 - \sim x = 1$


!0 = 1
decimal
~0 = 111111 ''
binary

Assignment Operators

$x \%= 10 \rightarrow x = x \% 10$

$x = 3$

$x /= 10 \Rightarrow x = x/10$

$x += 3 \rightarrow x = x+3$

$x *= 5 \rightarrow x = x*5$

= += -= *= /= %= &= ^= |= <<= >>=	Assignment operator Addition/subtraction assignment Multiplication/division assignment Modulus and bitwise assignment Bitwise exclusive/inclusive OR assignment	right to left

```
int a = b = 10;
```

```
v op= exp;
```

```
v = v op (exp);
```

```
v op= exp;

v = v op (exp);

x += y+1;

x = x + (y+1);
```

int x, y = 4

x = 5

x = (x * y) / 5 = 20/5 = 4 (wrong)

x *= y/5

x = x * (y/5) ⇒ 0

$x\ t = y/3$

$\Rightarrow \quad x = x + (y/3)$

$x\ /= y * 10$

$x = x / (y * 10)$

GO CLASSES

$a * = n+1$

$a = (a * n) + 1$ wrong

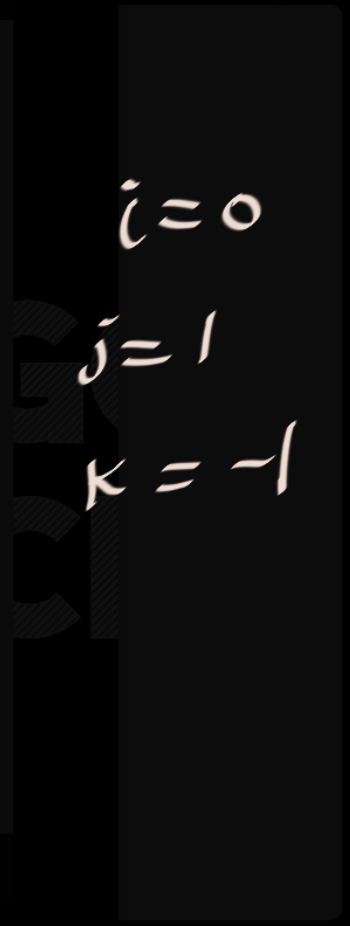
$a = a * (n+1)$ Right

Statement with simple assignment operator	*Statement with shorthand operator*
a = a + 1	a += 1
a = a – 1	a –= 1
a = a * (n+1)	a *= n+1
a = a / (n+1)	a /= n+1
a = a % b	a %= b

GATE CSE 2015

Consider the following C program:

```c
#include<stdio.h>
int main()
{
    int i, j, k = 0;
    j=2 * 3 / 4 + 2.0 / 5 + 8 / 5;
    k-=--j;
    for (i=0; i<5; i++)
    {
        switch(i+k)
        {
            case 1:
            case 2: printf("\n%d", i+k);
            case 3: printf("\n%d", i+k);
            default: printf("\n%d", i+k);
        }
    }
    return 0;
}
```



The number of times printf statement is executed is _____.

GATE CSE 2015

Consider the following C program:

```
for (i=0; i<5; i++)
{
    switch(i+k)
    {
        case 1:
        case 2: printf("\n%d", i+k);
        case 3: printf("\n%d", i+k);
        default: printf("\n%d", i+k);
    }
}
```

i: 0, 1, 2, 3, 4

i+k: -1, 0, 1, 2, 3

def. def.

i = 0

j = 1

k = -1

i+k: -1, 0, 1, 2, 3

1 Print 1 Print 3 3 2

10

The number of times printf statement is executed is _____.

GO CLASSES

```
int i, j, k = 0;
j=2 * 3 / 4 + 2.0 / 5 + 8 / 5;
```

6/4

= 1 + (2.0/5) + (8/5)

1 + 0.4 + 1 = 2.4


```
k-=--j;
```

k = k- --j ;

GATE CSE 2015

Consider the following C program:

```c
#include<stdio.h>
int main()
{
    int i, j, k = 0;
    j=2 * 3 / 4 + 2.0 / 5 + 8 / 5;
    k-=--j;
    for (i=0; i<5; i++)
    {
        switch(i+k)
        {
            case 1:
            case 2: printf("\n%d", i+k);
            case 3: printf("\n%d", i+k);
            default: printf("\n%d", i+k);
        }
    }
    return 0;
}
```

The number of times printf statement is executed is _____

i=0 , j=1, k=-1

Answer is **10**

Conditional Operator or ternary operator

```
exp1 ? exp2 : exp3
```

```
if (exp1)
{
   exp2
}
else   exp3
```

Conditional Operator

```
exp1 ? exp2 : exp3
```

```
5<2 ? 4 : 3;
```

= 3

```
int main()
{
    int a=0;
    a = 5<2 ? 4 : 3;
    printf("%d",a);

    return 0;
}
```

3

GATE CSE 2008

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. $x = 3, y = 4, z = 2$ → 4
B. $x = 6, y = 5, z = 3$ → 6
C. $x = 6, y = 3, z = 5$ → 6
D. $x = 5, y = 4, z = 5$ → 5

if (x>y)
    x>z ? x : z
else
    y>z : y : z

GATE CSE 2008

finding Max among 3 numbers

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  )
```

if (x>y)
  x>z ? x : z
else
  y>z : y : z

Comma Operator

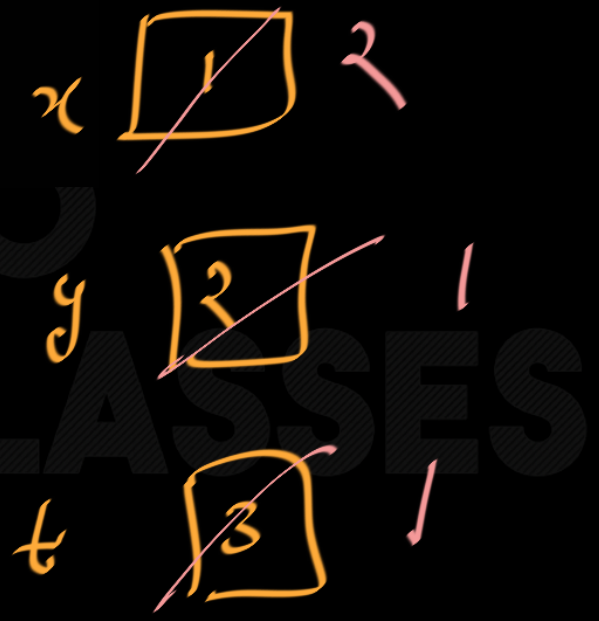
Evaluated left to right and the value of right-most expression is the value of the combined expression.

```
value = (x = 10, y = 5, x+y);
```

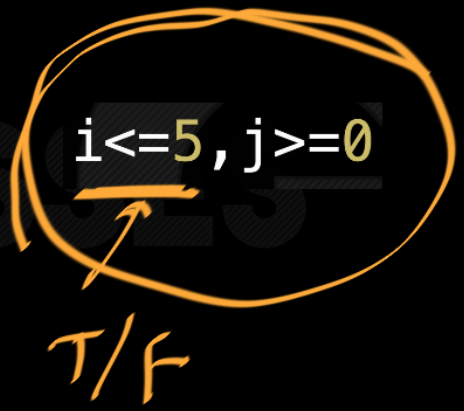
15

GO CLASSES

```
t = x, x = y, y = t;
```



```c
#include<stdio.h>

int main(){

    int i, j=2;
    for(i=0; i<=5,j>=0; i++)
    {
        printf("%d ", i+j);
        j--;
    }

    return 0;
}
```



```c
#include<stdio.h>

int main(){

    int i, j=2;
    for(i=0; i<=5,j>=0; i++)
    {
        printf("%d ", i+j);
        j--;
    }

    return 0;
}
```

i=0, 1, 2

j=2, 1, 0

i<=5,j>=0

T/F

```c
#include<stdio.h>

int main() {

    int i, j=2;
    for(i=0; j>=0,i<=5; i++)
    {
        printf("%d ", i+j);
        j--;
    }
    return 0;

}
```

Flip j>=0, i<=5

```c
#include<stdio.h>

int main() {

    int i, j=2;
    for(i=0; j>=0,i<=5; i++)
    {
        printf("%d ", i+j);
        j--;
    }
    return 0;

}
```

Flip j>=0, i<=5

i = 0, 1, 2, 3, 4, 5

```c
#include<stdio.h>

int main() {

    int i, j=2;
    for(i=0, j=2; j>=0 &&i<=5; i++, j--)
    {
        printf("%d ", i+j);

    }
    return 0;
}
```

very different from ";"

here BOTH j>=0 and i<=5 should be true

All operators with their associativity and precedence

Operator	Description	Associativity	Rank
() [] . -> ++ --	Parentheses or function call Brackets or array subscript Dot or Member selection operator Arrow operator Postfix increment/decrement	left to right	1
++ -- + - ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus and minus not operator and bitwise complement type cast Indirection or dereference operator Address of operator Determine size in bytes	right to left	2

* / %	Multiplication, division and modulus	left to right	3
+ -	Addition and subtraction	left to right	4
<< >>	Bitwise left shift and right shift	left to right	5
< <= > >=	relational less than/less than equal to relational greater than/greater than or equal to	left to right	6
== !=	Relational equal to or not equal to	left to right	7
&	Bitwise AND	left to right	8
^	Bitwise exclusive OR	left to right	9
|	Bitwise inclusive OR	left to right	10
&&	Logical AND	left to right	11
||	Logical OR	left to right	12
? :	Ternary operator	right to left	16

= += -= *= /= %= &= ^= |= <<= >>=	Assignment operator Addition/subtraction assignment Multiplication/division assignment Modulus and bitwise assignment Bitwise exclusive/inclusive OR assignment	right to left	14
,	comma operator	left to right	15

Highest and 2nd highest precedence operators

	Operators	Associativity
1	`()` `[]` `->` `.` `++` `--` (postfix)	left to right
2	`sizeof` `&` `*` `+` `-` `~` `!` *typecasts* `++` `--` (prefix)	right to left

Sequence points in C

GO CLASSES

```
a = 5 ;
```

```
x = a++ ;
```

after the semicolumn "a" get increemented.

Pf (x) ⟶ 5

Pf (a) ⟶ 6

GO CLASSES

```
a = 5 ;
```

```
x = a++ ;
```

after the Sequence point [illegible] "a" get increemented.

Pf (x) ⟶ 5

Pf (a) ⟶ 6

other Sequence points also exist. ⟶ Semicolomn is just one of the seq. point

GO CLASSES

$a = 5;$

$x = a++;$

Pf(x) → 5

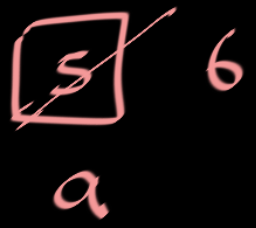
Pf(a) → 6

$x = a++ + a++;$

5

Pf(x)

Pf(a)



GO CLASSES

{ x = a++ + a++ ;

we are trying to modify the value of a multiple times before one sequence point

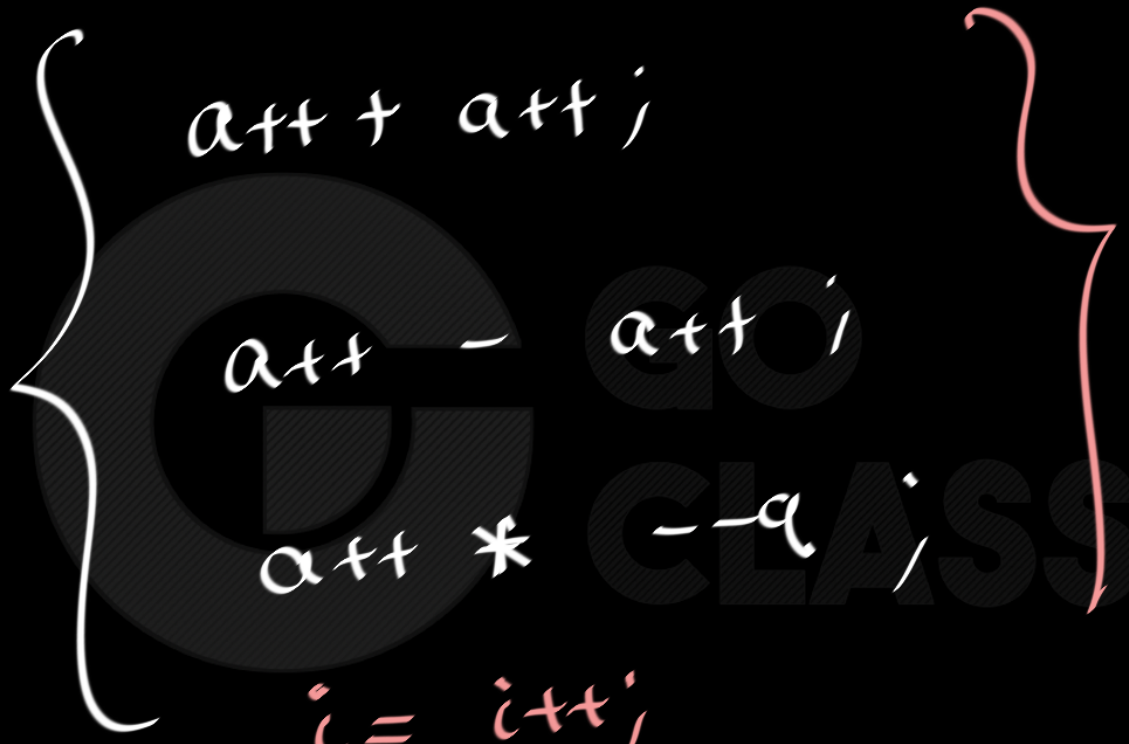
GO CLASSES

UNDEFINED

{ x = a++ + a++ i

is left to compiler.

we are trying to modify the value of a multiple times before one sequence point


a++ + a++ j
a++ - a++ j
a++ * --a j
i = i++ j
undefined
Compiler dependent

GO CLASSES

Sequence Points ← after that point the changes will definately take place.

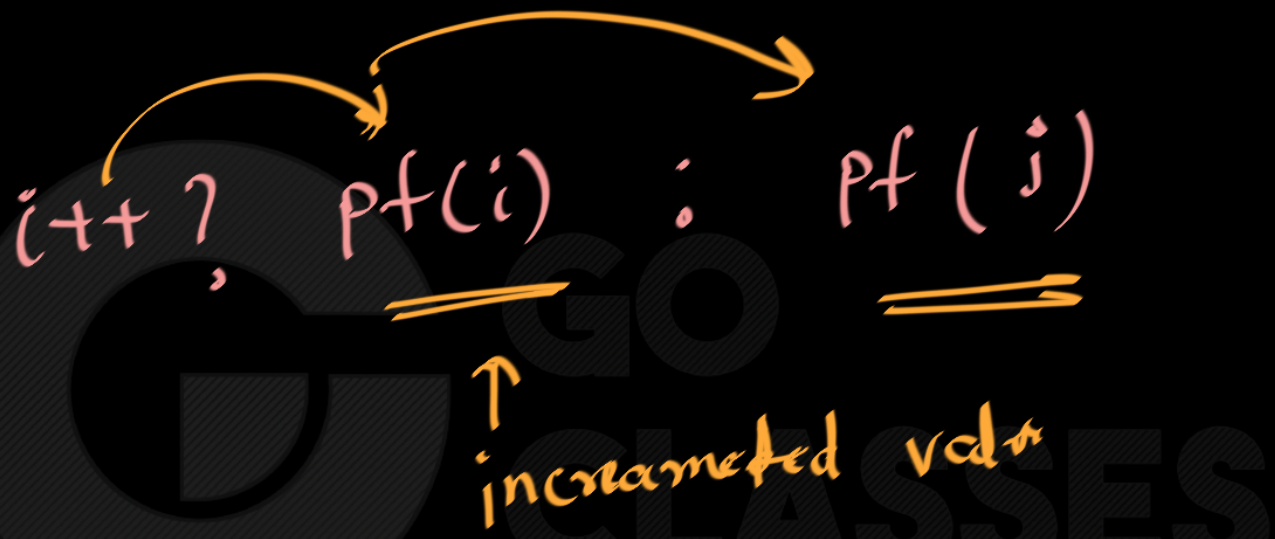
- → ;
- → if ( ) , for ( ) , while ( ) , switch
- → ? :
- → ||, &&

GO CLASSES

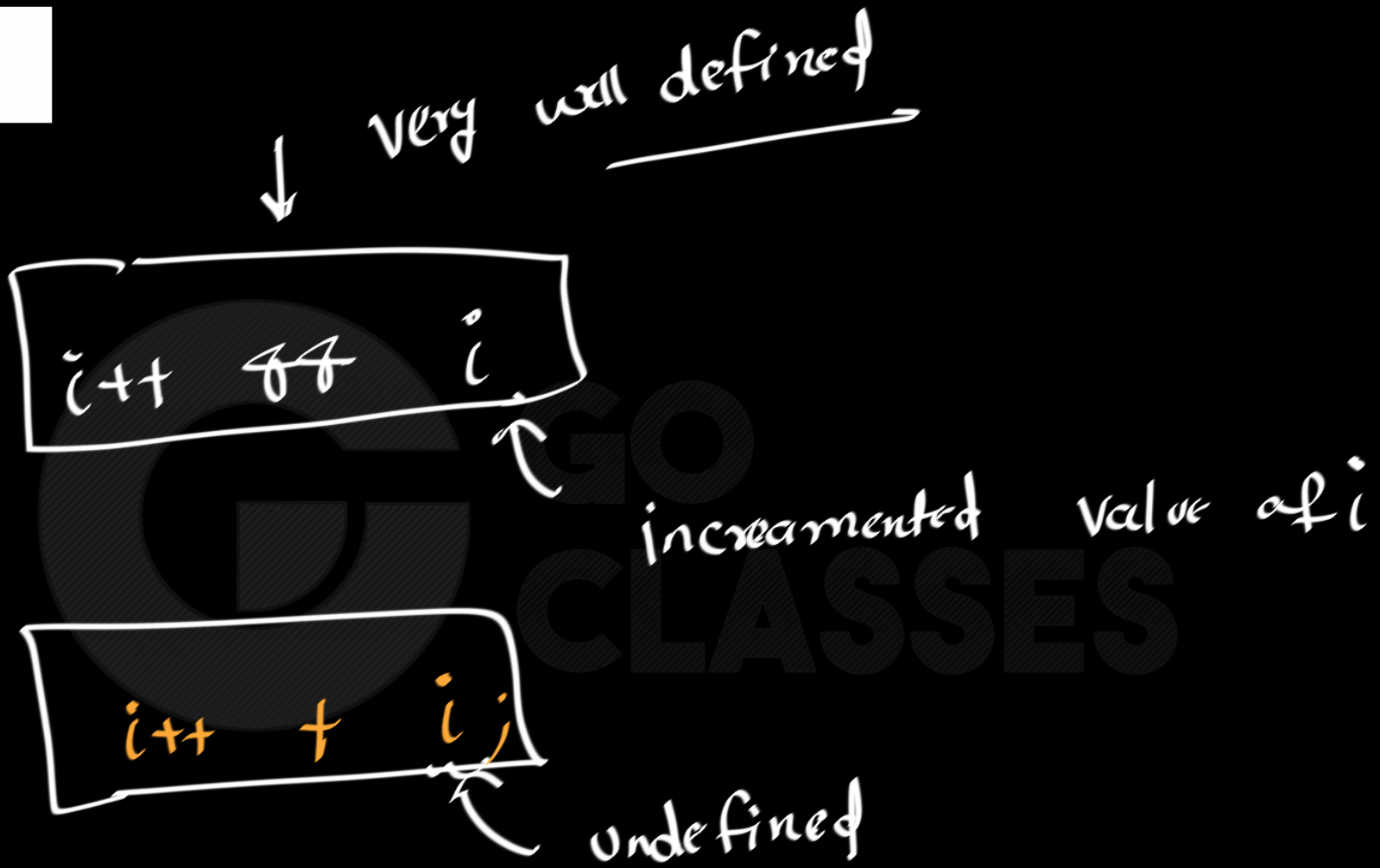
$i = 3$

```
if ( i++)
{
    pf (i)
}
```

Just after this i value will be 4.


i++ ? pf(i) : pf(j)
incremented val

GO CLASSES


very well defined
i++ ++ i
incremented value of i
i++ + i j
undefined

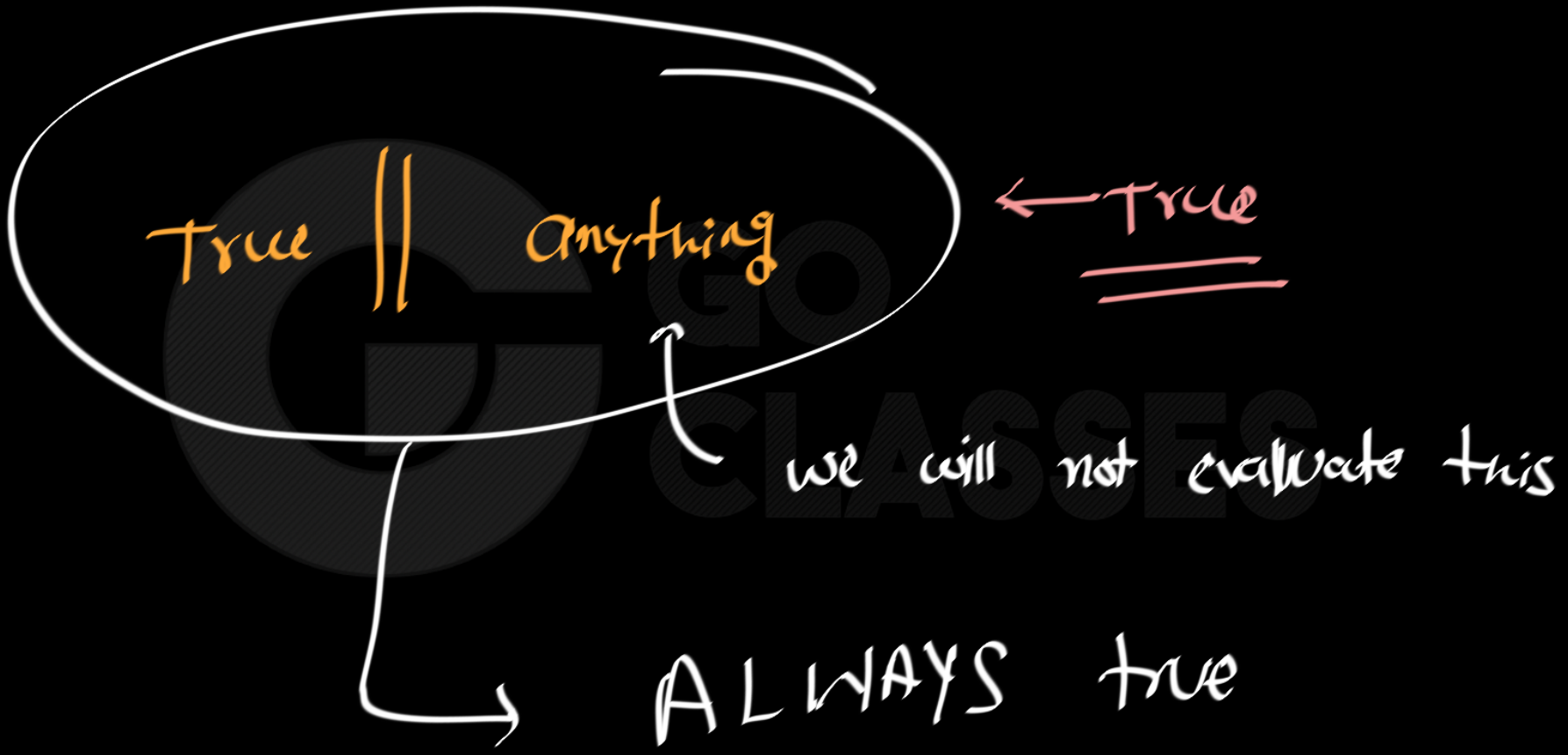
Short Circuiting in C

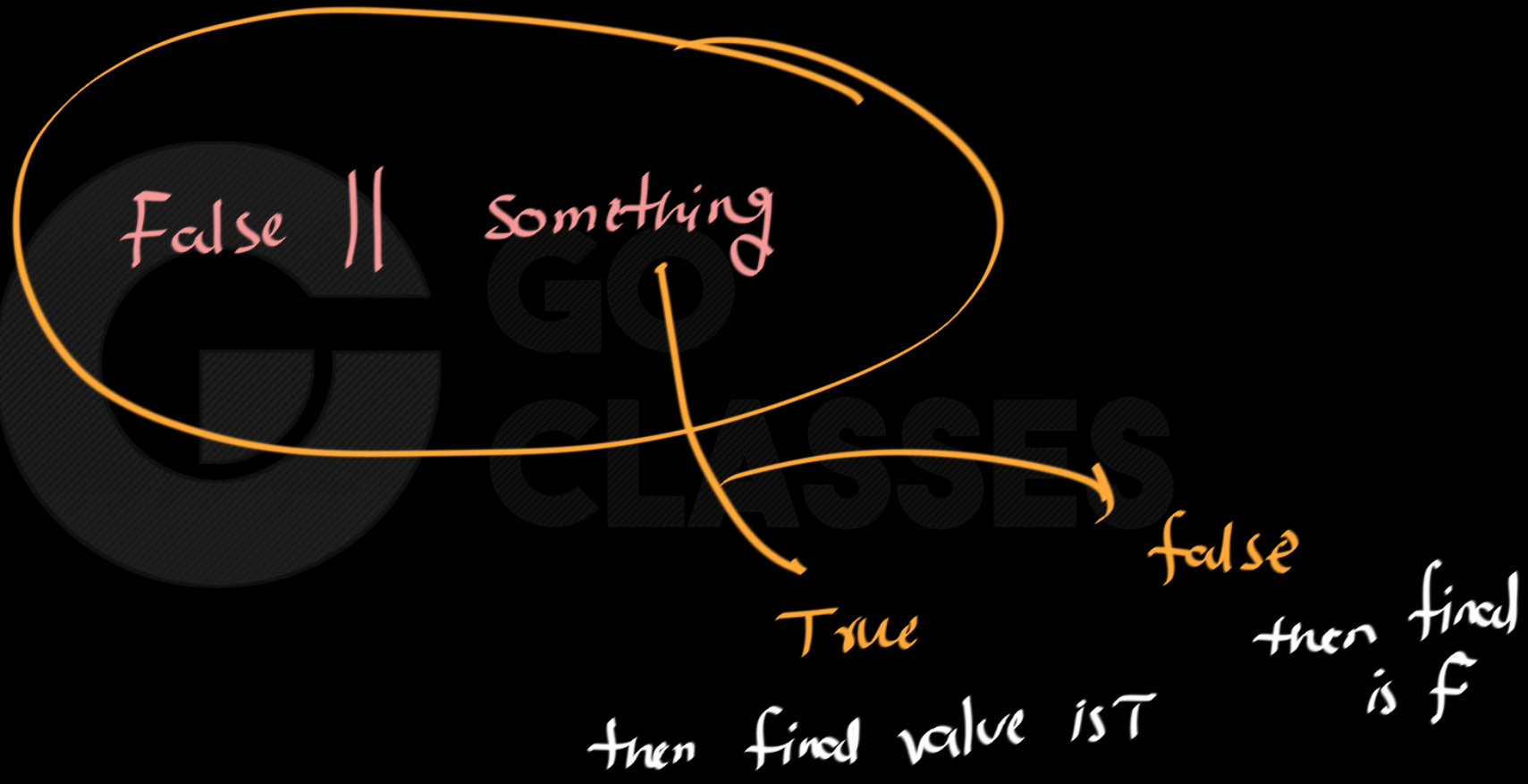
```c
#include <stdio.h>
int main()
{

    int a = 1, b = 1, c = 1;

    if (a == b || c++) {
        printf("%d", c);
    }

}
```

Not getting executed

```c
if (a == b || c++) {
    printf("%d", c);
}
```

true → TRUE || c++


True || anything
← True
we will not evaluate this
ALWAYS true


False || something
false
then final is F
True
then final value is T

GO CLASSES

```c
#include<stdio.h>

int main()
{
    int a=1, b=1,c=1;

    if (a != b || c++)
    {
        printf("%d ", c);
    }


}
```

will get executed

```
Desktop — -zsh — 80×24
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c -w
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
2 %
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

GO CLASSES

a.c

```c
#include<stdio.h>

int main()
{
    int a =1, b=1,c=1;

    if(a==b || c++){

        printf("%d\n", c);
    }
}
```

will not get executed

Desktop — -zsh — 80×24

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

GO CLASSES

true || anything ⇒ true

this will not get executed

GO CLASSES

Q: exp1 || exp2 || exp3

Put the brackets correctly

Answer: ((exp1 || exp2) || exp3)

Q: exp1 && exp2 && exp3

Put the brackets correctly

((exp1 && exp2) && exp3)

GO CLASSES

Q: exp1 && exp2 || exp3

Put the brackets correctly

Answer: ((exp1 && exp2) || exp3)

GO CLASSES

Q: exp1 || exp2 && exp3

Put the brackets correctly

Answer: (exp1 || (exp2 && exp3))

GO CLASSES

Q: exp1 && exp2 || exp3 && exp4

Put the brackets correctly

Answer : ((exp1 && exp2) || (exp3 && exp4))

GO CLASSES

Q: expr1 || expr2 && expr3 || expr4

Put the brackets correctly

Soln ((expr1 || (expr2 && expr3)) || expr4)

GO CLASSES

order of evaluation

exp1 || (exp2 && exp3)

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

    int a = 1, b = 1, c = 1;
    if (a == b || c++) {
        printf("%d", c);
    }

}
```

TRUE

don't care

1

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

    int a = 1, b = 1, c = 1;
    if (a != b || c++) {
        printf("%d", c);
    }

}
```

false

will get evaluated

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

    int a = 1, b = 1, c = 1;
          TRUE
    if (a == b && c++) {
        printf("%d", c);
    }

}
```

TRUE as C++

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

    int a = 1, b = 1, c = 1;
        false
    if (a != b && c++) {
        printf("%d", c);
    }
    printf("%d", c);
}
```

GO CLASSES

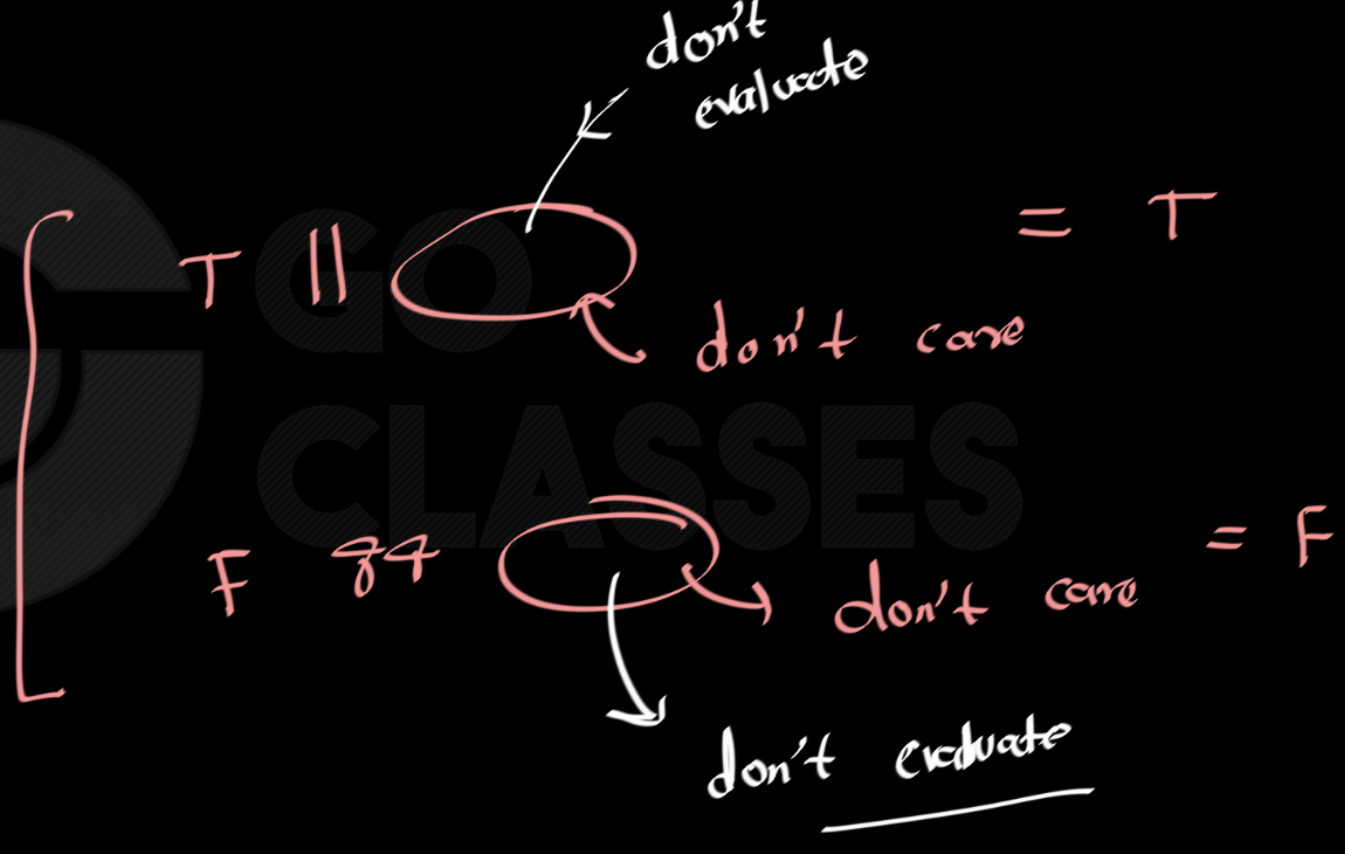
```c
#include<stdio.h>

int main()
{
    int a=1, b=1,c=1;

    if (a != b && c++)
    {
        printf("%d ", c);
    }

    printf("%d ", c);


}
```

false ← no evaluation

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

GO CLASSES

SHORT CIRCUIT BEHAVIOUR



GO CLASSES

Order of evaluation

```
int a = 1 ;
int b = a+1 ;
```

will execute this first

Order of evaluation

GO CLASSES

Order of evaluation in logical && and || then it is ALWAYS Left to right

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

    int a = 1, b = 2, c = 1;

    if (a != b || c++) {
        printf("%d", c);
    }

}
```

TRUE

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

    int a = 1, b = 2, c = 1;

    if (a != b || c++) {
        printf("%d", c);
    }

}
```

TRUE

No evaluation

GATE 2021

```c
#include <stdio.h>

int main()
{
    int i, j, count;
    count=0;
    i=0;
    for (j=-3; j<=3; j++)
    {
        if (( j >= 0) && (i++))
            count = count + j;
    }
    count = count +i;
    printf("%d", count);
    return 0;
}
```

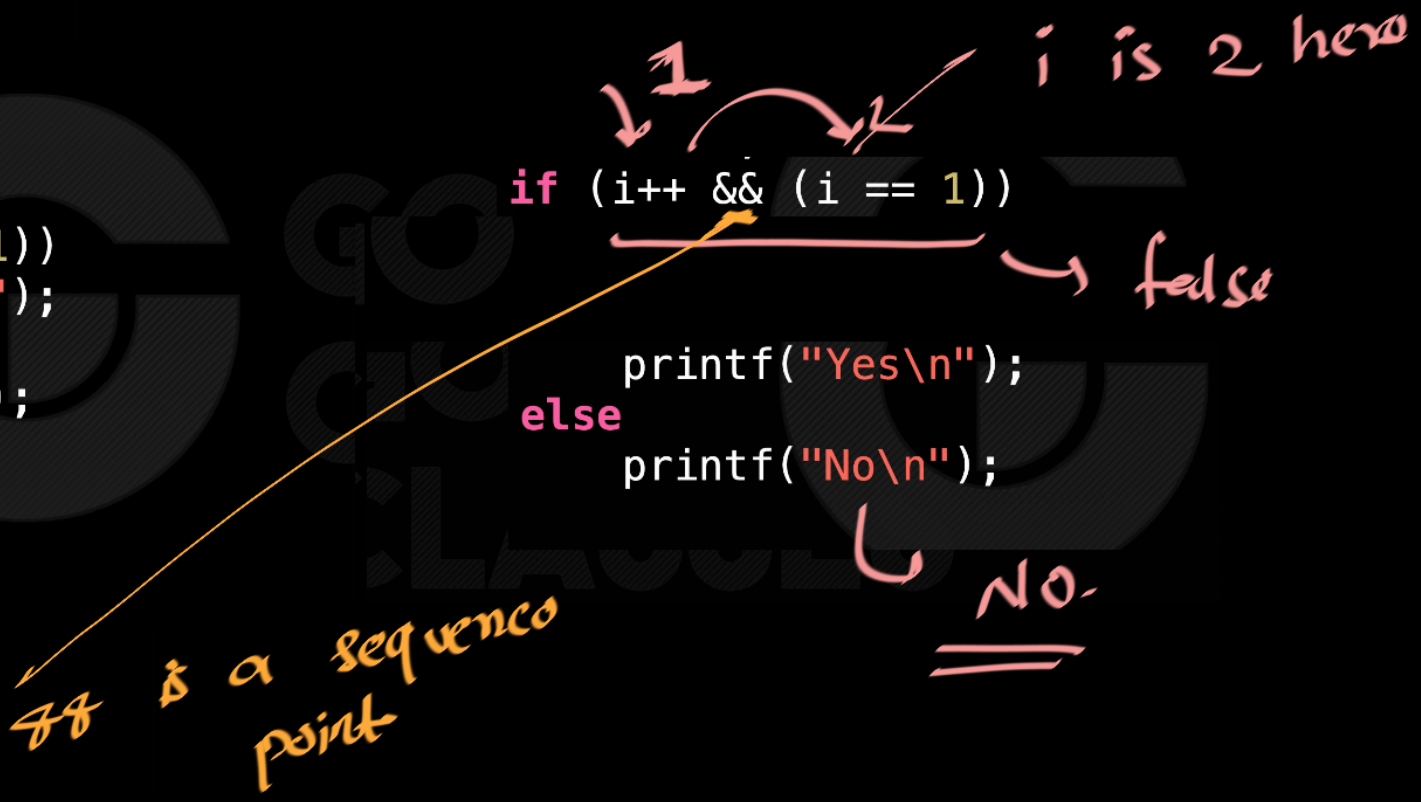
H.W.

Which one of the following options is correct?

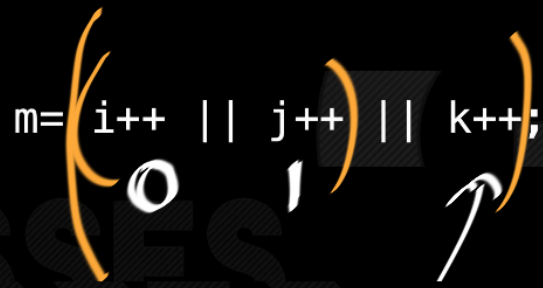
A. The program will not compile successfully
B. The program will compile successfully and output 10 when executed
C. The program will compile successfully and output 8 when executed
D. The program will compile successfully and output 13 when executed

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


{
    int i = 1;
    if (i++ && (i == 1))
        printf("Yes\n");
    else
        printf("No\n");
}
```

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

{
    int i = 1;
    if (i++ && (i == 1))
        printf("Yes\n");
    else
        printf("No\n");
}
```



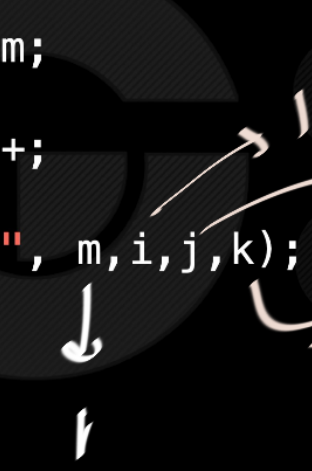
```c
#include<stdio.h>
main()

{

    int i=0, j=1, k=2, m;

    m= i++ || j++ || k++;

    printf("%d %d %d %d", m,i,j,k);

}
```



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

{

    int i=0, j=1, k=2, m;

    m= i++ || j++ || k++;

    printf("%d %d %d %d", m,i,j,k);

}
```

i=1

j=2



bcoz expression is TRUE

No evaluation

GO CLASSES

a.c

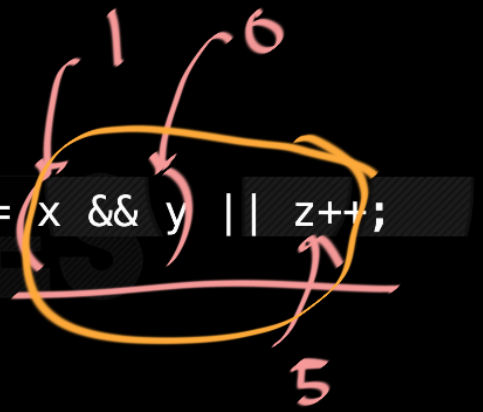
```c
#include<stdio.h>

int main()
{
    int i = 0, j =1, k=2, m;

    m = i++ || j++ || k++;


    printf("%d %d %d %d ", m,i,j,k);


}

```

```
Desktop — -zsh — 80×24
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c -w
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
1 1 2 2 %
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

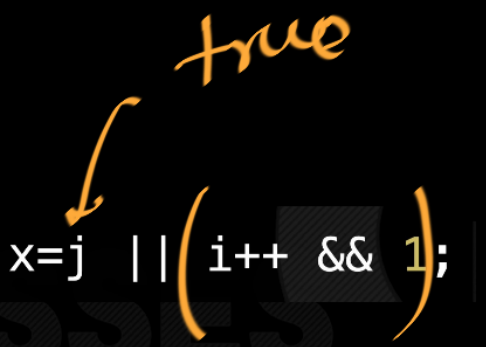
```c
#include <stdio.h>
    void main()
    {
        int x = 1, y = 0, z = 5;

        int a = x && y || z++;

        printf("%d", z);
    }
```

```c
int a =(x && y) || z++;
```

```c
#include <stdio.h>
    void main()
    {
        int x = 1, y = 0, z = 5;

        int a = x && y || z++;

        printf("%d", z);
    }
```



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

    int x,i=4,j=7;

    x=j || i++ && 1;

    printf("%d",i);
}
```



```c
#include<stdio.h>
int main()
{
    int i=-3, j=2, k=0, m;
    m = ++i && ++j && ++k;
    printf("%d, %d, %d, %d\n", i, j, k, m);
    return 0;
}
```

H.W.

A. -2, 3, 1, 1
B. 2, 3, 1, 2
C. 1, 2, 3, 1
D. 3, 3, 1, 2

```c
#include<stdio.h>
int main()
{
    int i=-3, j=2, k=0, m;
    m = ++i || ++j && ++k;
    printf("%d, %d, %d, %d\n", i, j, k,m);
    return 0;
}
```

H.W

A. 2, 2, 0, 1
B. 1, 2, 1, 0
C. -2, 2, 0, 0
D. -2, 2, 0, 1

```c
#include<stdio.h>
void main()
{
int k,i=50,j=100,l;


i=i|(j&&100);


k=i||(j||100);


l=i&(j&&100);


printf("%d %d\n",i,j);
printf("%d %d",k,l);
}
```

H.W.

GO CLASSES

true || ← don't evaluate

false && ← don't evaluate

} Short Circuiting

{ You always go from left to right evaluation }

put the brackets based on Assoc. and preced. ↑ order of evaluation

Step1 : put the brackets

Step2 : go from left to right

Step3 : See if the Short Circuits are there

(T || ⊙ , F && ⊙)