



Operators in C programming



Operator

- Arithmetic Operators
- Relational Operators
- Logical Operators

.....



GO
CLASSES



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.



C Programming

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

$3 + 2 * 5$

$$\underline{\underline{2 * 3}} / 6 = 1$$

$$2 * (8/6) = 0 \times$$



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 = 10
```



C Programming

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





C Programming

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

* , / , %
↓
First pass

Step1: x = 9-4+3*2-1

Step2: x = 9-4+6-1

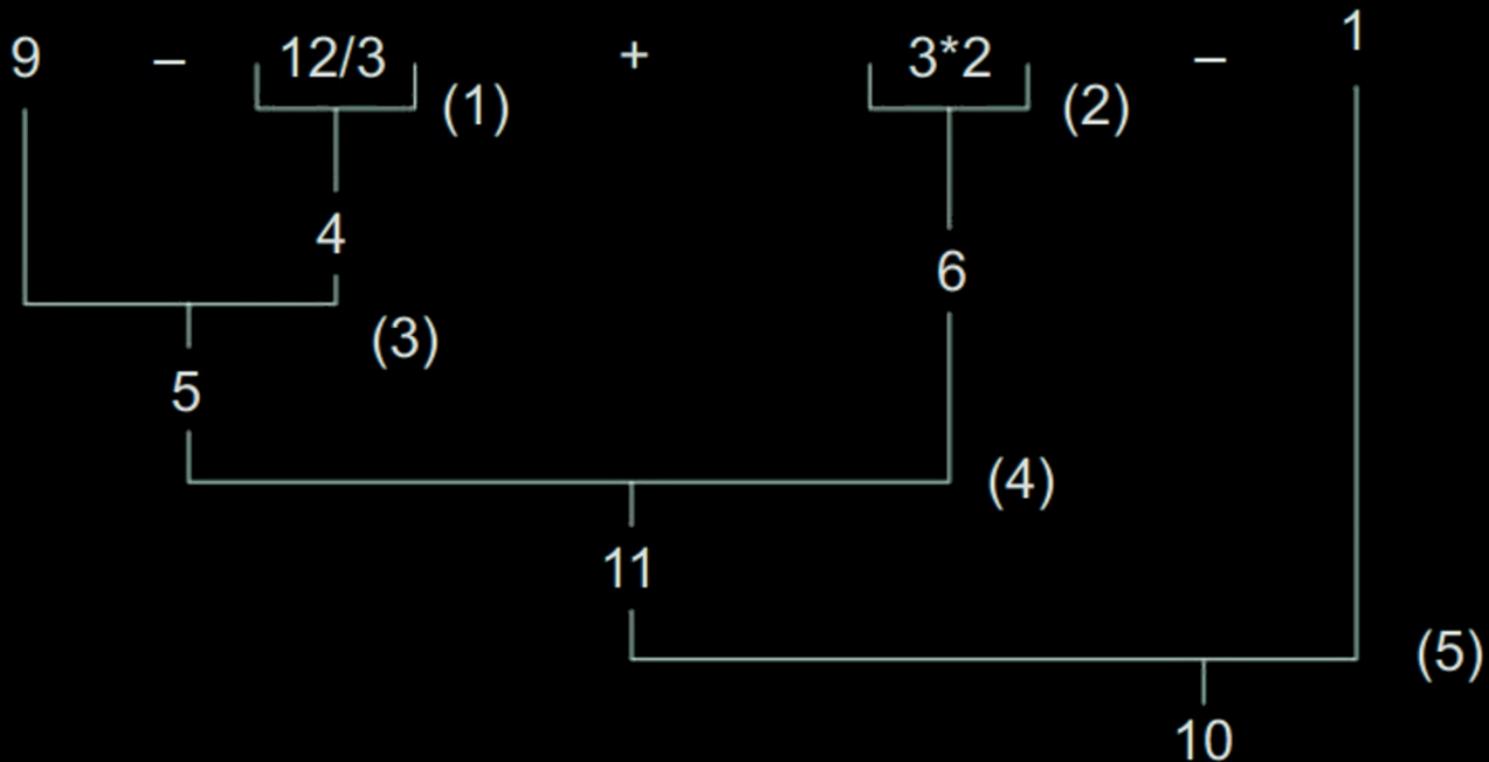


Second pass

Step3: x = 5+6-1

Step4: x = 11-1

Step5: x = 10





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



C Programming

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

Question 1

```
#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
practices → { a < b && b < c }

Question 2

```
#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;
}
```

$$(30 > 20) > 10$$

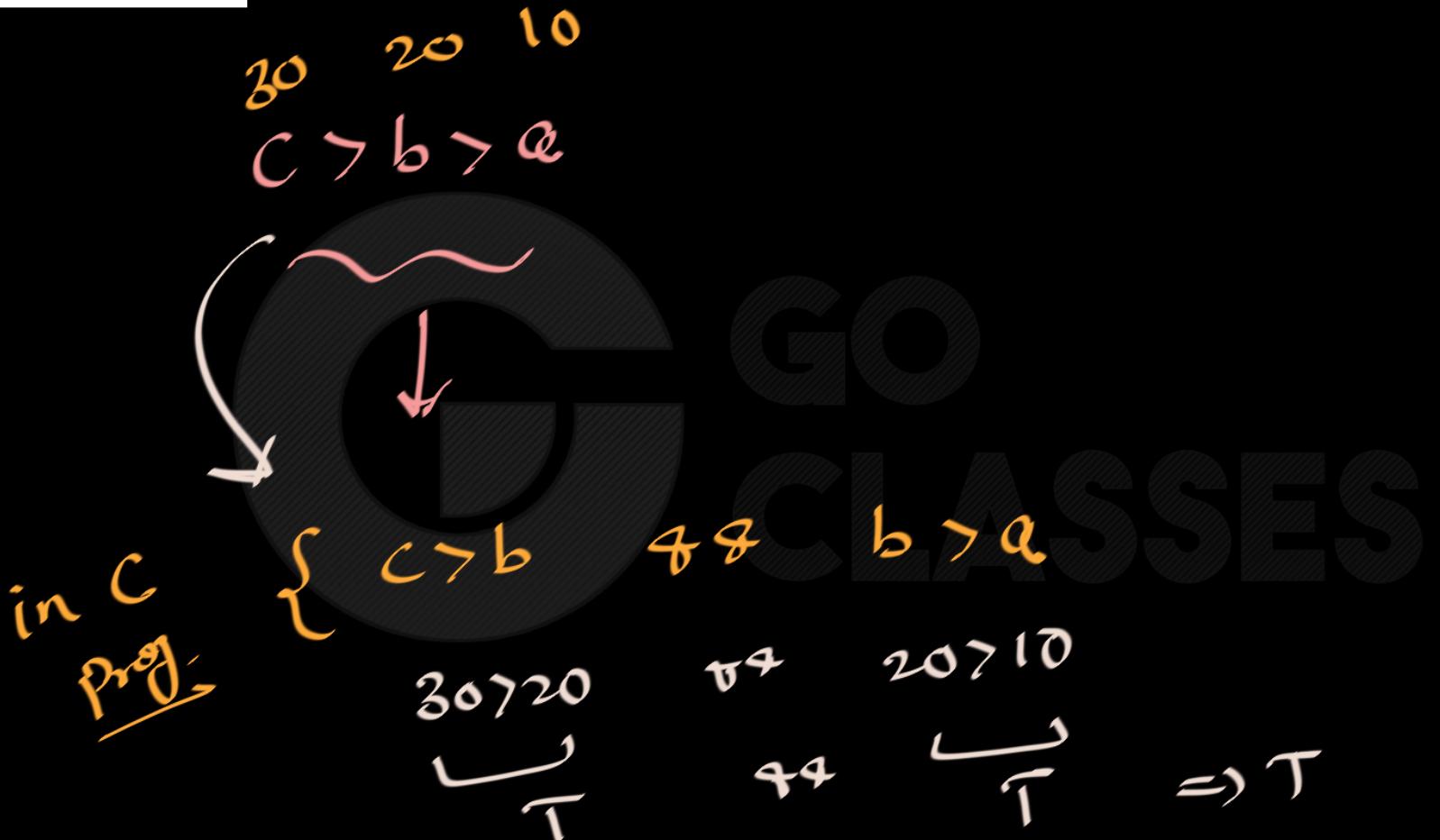
True

$$\equiv 1 > 10$$

↓
false

False

Answer





LOGICAL OPERATORS

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





C Programming

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



C Programming

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

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

    return 0;
}
```



l < 10 && 2 > 0
↓ ↓
True True

↓
True



C Programming

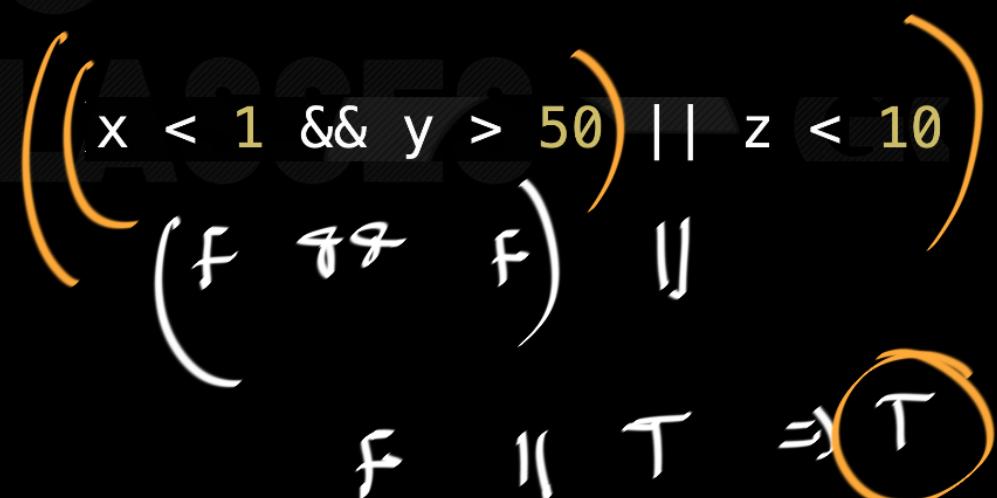
```
int x = 10, y = 20, z = 1;  
  
if (x < 1 && y > 50 || z < 10)  
    printf("True");  
else  
    printf("False");
```



&&	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");
```

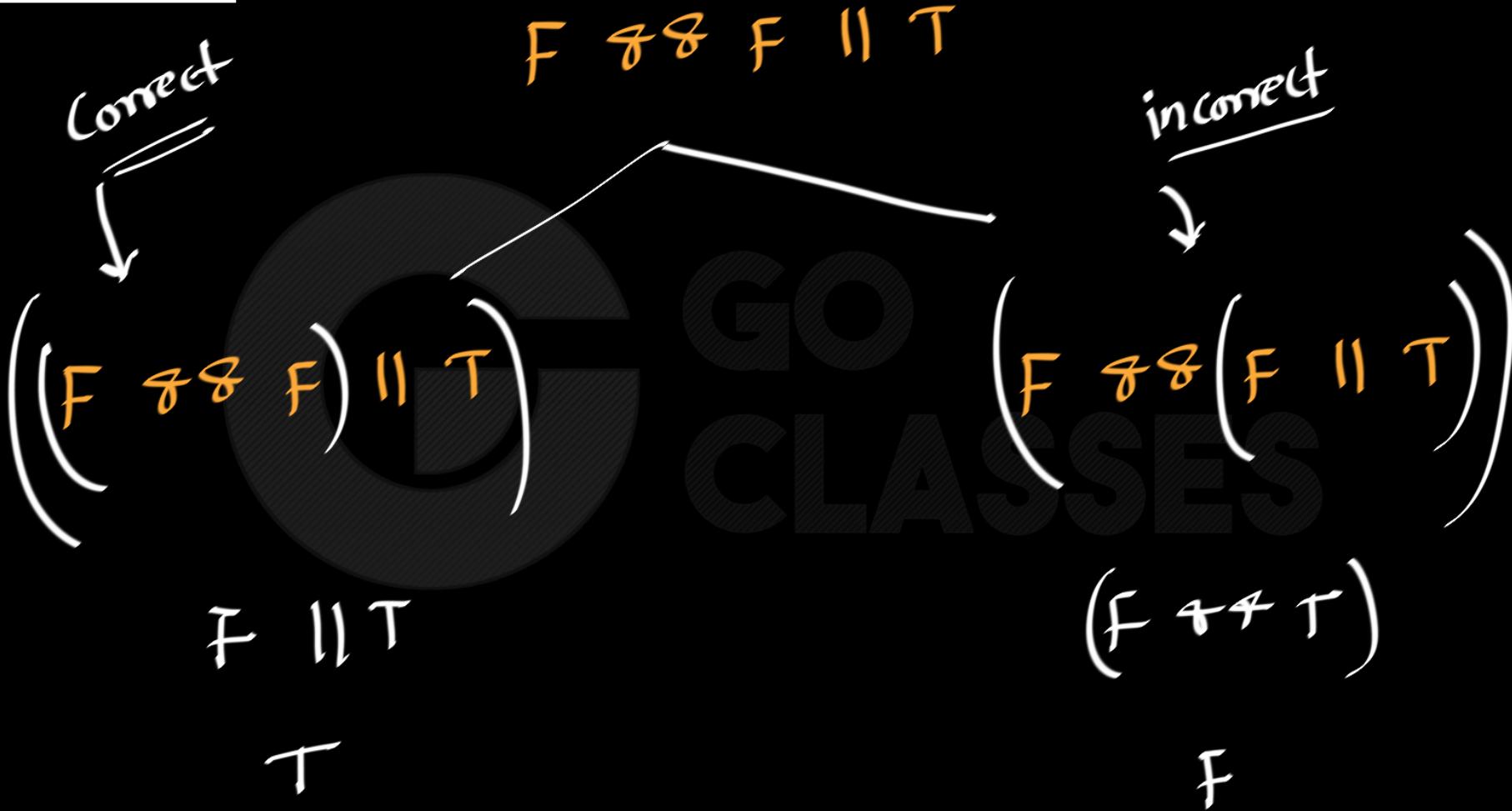


By mistake

$$\overline{F} \quad x < 1 \quad \&& \left(\overline{F} \quad y > 50 \quad || \quad z \overline{T} < 10 \right)$$

$$F \quad \&& \left(F \quad || \quad T \right)$$

$$F \quad \&& \quad T \quad \Rightarrow F$$



Q. $\exp^1 \parallel \exp^2 \parallel \exp^3$

put the brackets correctly

Answer

$$\left(\exp^1 \parallel \exp^2 \right) \parallel \exp^3$$

Q. $\exp(1 + 4t + \exp^2) + 8t \exp^3$

put the brackets correctly

$$\left(\left(\exp(1 + 4t + \exp^2) \right) + 8t \exp^3 \right) \checkmark$$

Q: $\text{exp}^1 \neq \text{exp}^2 \parallel \text{exp}^3$

put the brackets correctly

Answer

$$((\text{exp}^1 \neq \text{exp}^2) \parallel \text{exp}^3)$$

Q: $\text{expr1} \parallel \text{expr2} + \text{expr3}$
Put the brackets correctly

Answer

$$(\text{expr1} \parallel (\text{expr2} + \text{expr3}))$$

Q:

exp1 44 exp2 || exp3 88 exp4

put the brackets correctly

Answer:
$$\left(\text{exp1} \quad 44 \text{ exp2} \right) \parallel \left(\text{exp3} \quad 88 \text{ exp4} \right)$$

Q:

 $\text{expr1} \parallel \text{expr2} \& \text{expr3} \parallel \text{expr4}$

Put the brackets correctly

Soln
$$\left(\text{expr1} \parallel \left(\text{expr2} \& \text{expr3} \right) \right) \parallel \text{expr4}$$



order of evaluation

expr || (expr & & expr)

CLASSES



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;`)



C Programming

n = 5;

x = n++;

x = ++n;



x = n ;
n = n+1 ;

int main()

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



```
int main( )
```

```
{
```

```
    int n = 5;
```

```
    int x;
```

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

pf(x) → 5

pf(n) → 6

```
}
```

post - increment



```
int main( )
```

```
{
```

```
    int n = 5;
```

```
    int x;
```

```
x = +tn;
```

```
pf(x) → 6
```

```
pf(n) → 6
```

```
}
```

Pre - increment



C Programming

Illegal operations

$\text{++}2 \leftarrow \text{constant}$

$\text{++}(a+b^*2)$



Bitwise Operators

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

GO
CLASSES



C Programming

x y

$op1$	$op2$	$op1 \& op2$	$op1 op2$	$op1 ^ op2$
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

$x \oplus y$

$(\alpha + \beta)$ ^{operator}
α operand
β operand

$x \oplus y$



Bitwise And : &

x - - -> 0000 0000 0000 1101 → 13

y - - -> 0000 0000 0001 1001 → 25

z = x & y ;

z - - -> 0000 0000 0000 1001

3 2 1 0
↓ ↓ ↓ ↓
1 1 0 1

↑
24

iSES

16
9
—
25

9



The image shows a screenshot of a terminal window titled 'c a.c'. The terminal is running on a Mac OS X desktop, as indicated by the window title bar. The command entered is 'Desktop -- zsh -- 80x24'. The output of the program is displayed, showing the value '9' printed to the screen.

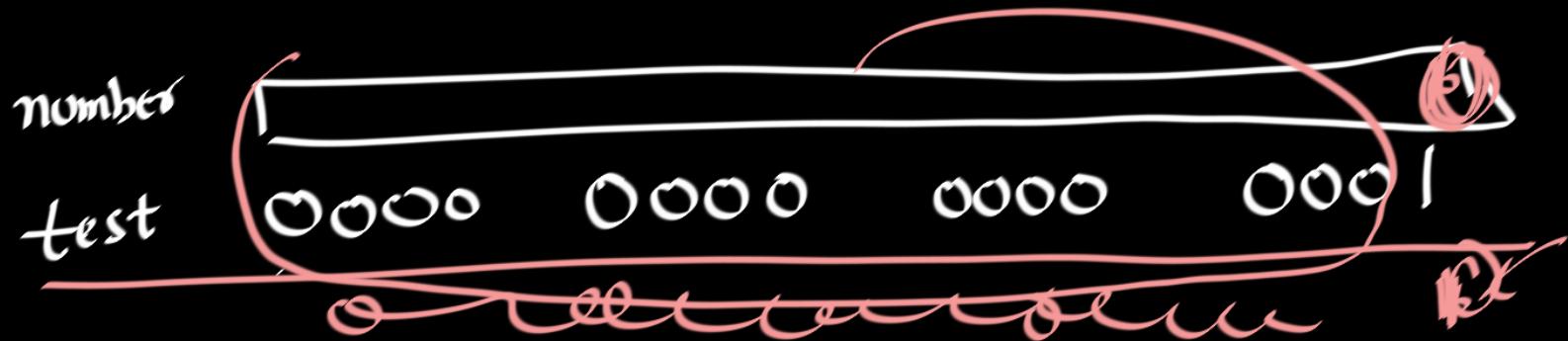
```
#include<stdio.h>
int main() {
    int x = 13;
    int y = 25;
    int z;
    z = x&y;
    printf("%d\n", z);
    return 0;
}
```



C Programming

```
int test = 1;  
  
if(number & test)  
    print("Number is odd\n\n");  
else  
    printf("Number is even\n\n");
```

SES





Bitwise OR : |

x - - ->	0000 0000 0000 1101	→ 13
y - - ->	0000 0000 0001 1001	→ 25
x y - - ->	0000 0000 0001 1101	→ 29

Binary representation of 13: 0000 0000 0000 1101
Binary representation of 25: 0000 0000 0001 1001
Result of x|y: 0000 0000 0001 1101 → 29

Annotations:

- Downward arrows from the binary digits to their corresponding powers of 2:
 - 1st digit: 2^0
 - 2nd digit: 2^1
 - 3rd digit: 2^2
 - 4th digit: 2^3
 - 5th digit: 2^4
 - 6th digit: 2^5
 - 7th digit: 2^6
 - 8th digit: 2^7
- A bracket below the binary digits groups them into pairs: (0000), (0000), (0001), (1101).



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:

$op \ll n$

Right shift:

$op \gg n$



Left shift

suppose x is 0100 1001 1100 1011

x << 3 = 0100 1110 0101 1000

$x =$

0100 1001 1100 1011

 $x \ll 3 =$

0 1001 1100 1011000

ES

$x = 3$ $\sum_{x=2}^{x=2}$

$$\frac{4}{12}$$

3

$$\frac{001100}{12}$$



GO CLASSES



Right Shift

suppose x is 0100 1001 1100 1011

x >> 3 = 0000 1001 0011 1001



Right Shift

suppose x is ~~0100 1001 1100 1011~~
 $x \gg 3 = \underline{0000} \ 1001 \ 0011 \ 1001$



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 Programming

```
#include<stdio.h>
int main(){
    int x,y =10;
    x = y << 1;
    printf("%d",x);
}
```

↳ 20

```
#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

$\sim x$ = 0110 1001 0011 0100

S

! 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 */
              ^~~~~~ ^~~~~~ ^~~~~~ } sign
    printf("%d", ~x);
}
```

-16+7 = -9

$$-\bar{x} = \bar{\bar{x}} + 1 \quad (\text{in } 2's \text{ Complemented System})$$


$$\begin{array}{rcl} \overbrace{\bar{x} + 1}^{\text{2's comp}} & - & \bar{x} \\ \downarrow & & \downarrow \\ -\bar{x} & = & 1 \end{array}$$



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

A handwritten calculation is shown below the code. It starts with $\sim x$ and $\sim \sim x$, both of which are crossed out with a large red arrow pointing from the original code line. To the right of these crossed-out terms is an equals sign (=), followed by the number 1, indicating the result of the expression $-x - \sim \sim x$.

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

```
(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 %
```

10 = 1 *decimal*

~0 = 11111111 *binary*



Assignment Operators

 $x = 3$ $x += 3$ $x *= 5 \rightarrow x = x * 5$ $x \&= 10 \rightarrow x = \underline{x \& 10}$ $x /= 10 \rightarrow$ $x = x / 10$ 



C Programming

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

right to left

$x = 3$



C Programming

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





C Programming

v op= exp;

v = v op (exp);





C Programming

v op= exp;

int x, y = 4

x=5

v = v op (exp);

$$x = \underbrace{(x * y)}_5 = \frac{20}{5}$$

x += y+1;

$$x *= y / 5$$

x = x + (y+1);

$$x = x * \left(\frac{y}{5}\right) \Rightarrow 0$$

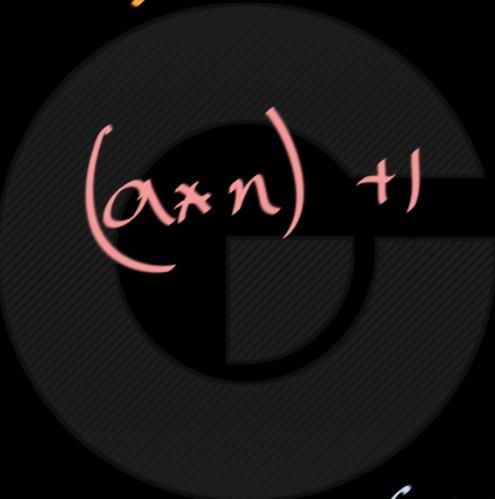
wrong

$$x_t = \frac{y}{3}$$
$$\Rightarrow x = x_t \left(\frac{y}{3} \right)$$

$$x_t = \frac{y * 10}{3}$$
$$x = \frac{x_t * 10}{3}$$

$$a * = n + 1$$

a =



wrong

GO
CLASSES

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

Right

=====

✓



C Programming

<i>Statement with simple assignment operator</i>	<i>Statement with shorthand operator</i>
<code>a = a + 1</code>	<code>a += 1</code>
<code>a = a - 1</code>	<code>a -= 1</code>
<code>a = a * (n+1)</code>	<code>a *= n+1</code>
<code>a = a / (n+1)</code>	<code>a /= n+1</code>
<code>a = a % b</code>	<code>a %= b</code>

Consider the following C program:

```
#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;
}
```

$$i = 0$$

$$j = 1$$

$$k = -1$$

The number of times printf statement is _____.

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);
    }
}
```

The number of times printf statement is _____.

→ 10

$i = 0, 1, 2, 3, 4$
 $i+k = -1, 0, 1, 2, 3$

def. def.

$i = 0$
 $j = 1$
 $k = -1$

$i+k = -1, 0, 1, 2, 3$

↓ ↓ ↓ ↓ ↓
 1 print 1 print 3 3 2

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

$6/4$

$$= 1 + \cancel{\left(0/5 \right)} + \left(0/5 \right)$$

$$\boxed{1 + 0 + 1}$$

`k=--j;`

$$k = k - \cancel{--j} j$$

$\boxed{0}$
i

~~$\boxed{0}$~~ 3
j

~~$\boxed{0}$~~ -1
k

Consider the following C program:

```
#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;
}
```

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

SESSES

Answer is 10

The number of times printf statement is _____



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?

$$a = (x > y) ? \left(\underbrace{(x > z) ? x : z} \right) : \left(\underbrace{(y > z) ? y : z} \right)$$

- A. $x = 3, y = 4, z = 2 \rightarrow 4$
- B. $x = 6, y = 5, z = 3 \rightarrow 6$
- C. $x = 6, y = 3, z = 5 \rightarrow 6$
- D. $x = 5, y = 4, z = 5 \rightarrow 5$

if ($x > y$)

$x > z ? x : z$

else $y > z : y : z$

$y > z$



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?

• z

• x

• y

• x

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

• z

• y

• x

if ($x > y$)

$x > z ? x : z$

else $y > z : y : z$

• x

(y, z)

• x



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
value

10
x

5
y

$t = x, x = y, y = t;$

$x \sqrt{1}^2$

$y \sqrt{2}^1$

$t \sqrt{3}^1$



```
#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<=5, j>=0
T/F

```
#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 Programming

```
#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 Programming

```
#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

(=0, 1, 2, 3, 4, 5)



C Programming

```
#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 <u>Unary plus and minus</u> not operator and bitwise complement type cast Indirection or dereference operator Address of operator Determine size in bytes	right to left	2

<i>Arith</i>	*	/	%	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
<i>Relation</i>	<	<=		relational less than/less than equal to	left to right	6
	>	>=		relational greater than/greater than or equal to	left to right	
	==	!=		Relational equal to or not equal to	left to right	7
<i>Bitwise</i>	&			Bitwise AND	left to right	8
	^			Bitwise exclusive OR	left to right	9
				Bitwise inclusive OR	left to right	10
<i>logical</i>	&&			Logical AND	left to right	11
				Logical OR	left to right	12
	? :			Ternary operator	right to left	13



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



Highest and 2nd highest precedence operators



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



C Programming



Sequence points in C

$\alpha = 5 ;$ $x = \alpha ++ ;$ $pf(x) \rightarrow 5$ $pf(\alpha) \rightarrow 6$

after the semicolon

“ α ” get incremented.

$\alpha = 5 ;$ $x = \alpha ++ ;$ $\text{Pf}(x) \rightarrow 5$ $\text{Pf}(\alpha) \rightarrow 6$

other Sequence points also exist.

after the sequence point

' α ' get incremented.

Semicolon is just
one at the seq-
point

$\alpha = 5 ;$ $x = \alpha ++ ;$ $Pf(x) \rightarrow 5$ $Pf(\alpha) \rightarrow 6$ $x = \alpha + \cancel{\alpha} + \underline{\alpha + j} \downarrow$ $Pf(x)$ $Pf(\alpha)$  6
 α

{ $x = a + t + a + t ;$

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

UNDEFINED

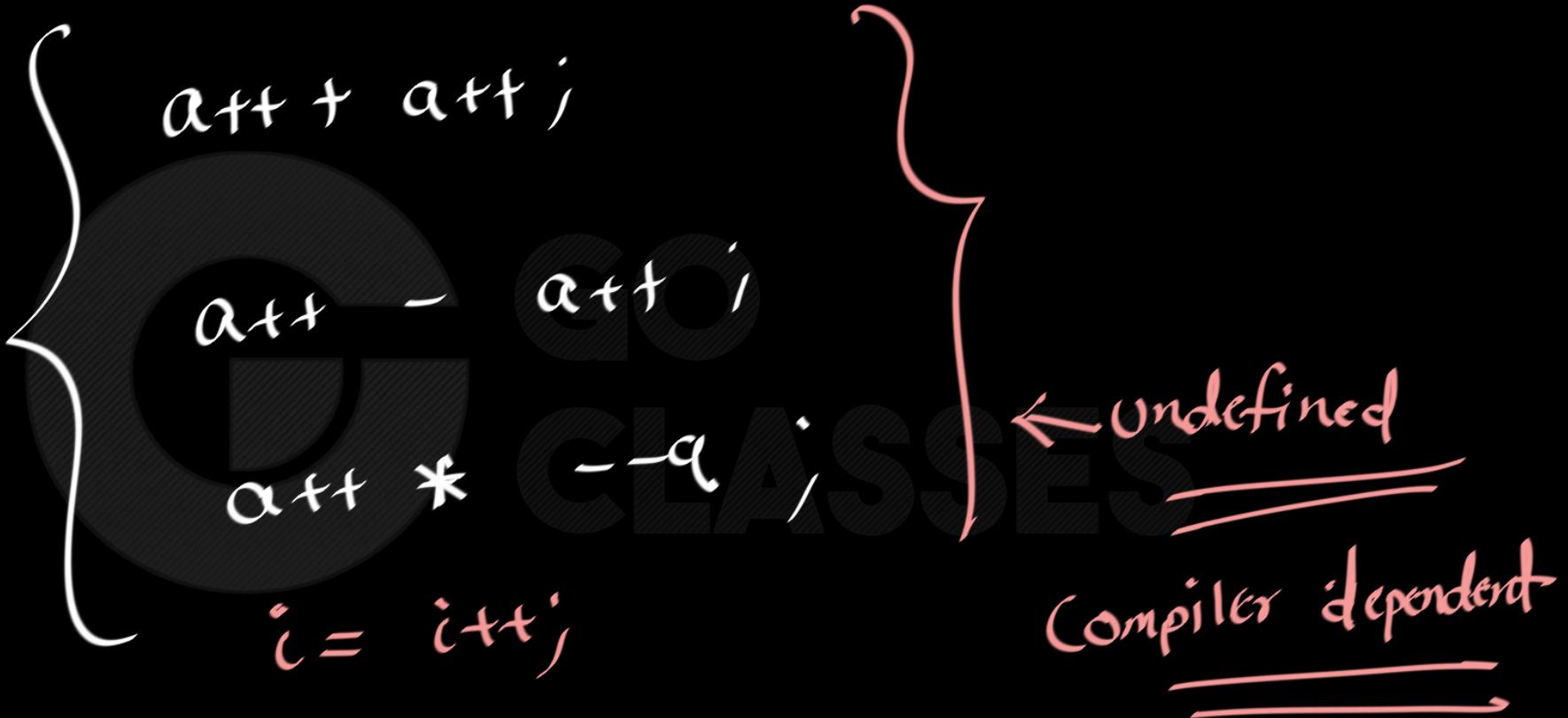
{ $x = a++ + a++ ;$ } is left to compiler.
we are trying to modify the value of
a multiple times before one sequence point

++ + ++ ; ++ / ; = ++ ;

++ - = -

i = i++ ;

undefined
Compiler dependent



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

- ;
- if (), for (), while (), switch
- ?, :
- {}, &

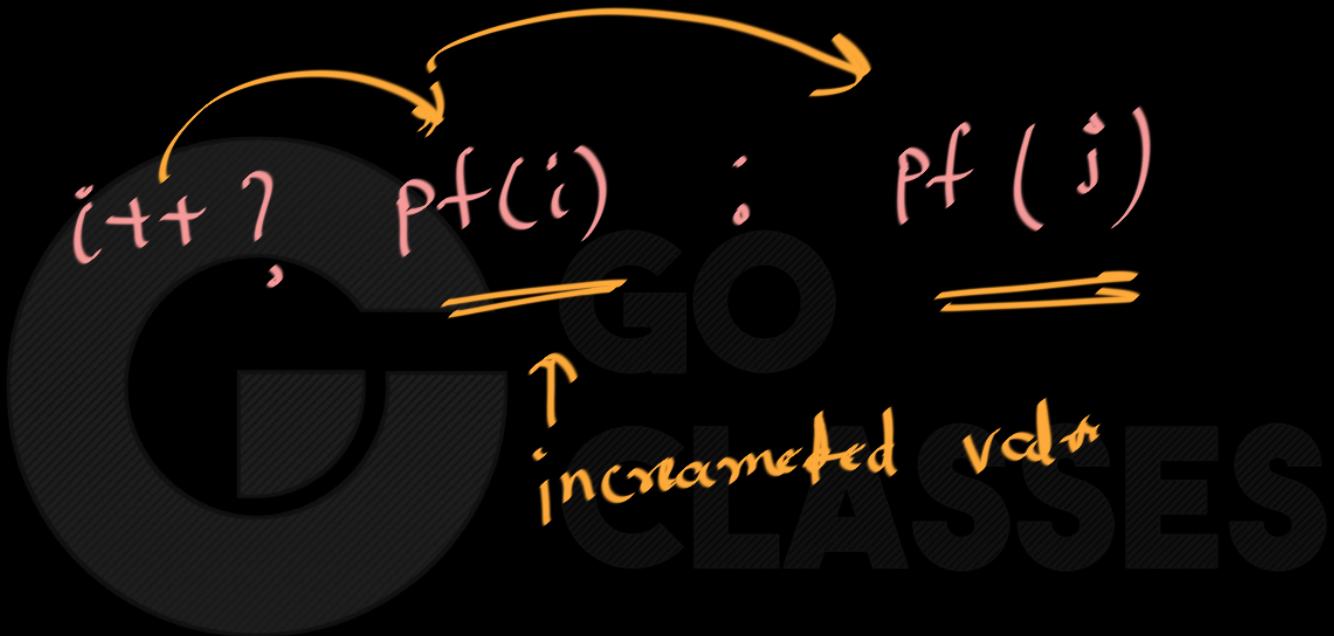
$i = 3$

if ($i++$)
just after this i value
will be 4.

{

pt(i)

}





int *ff* = 0;
for (*i* = 0; *i* < 10; *i*++) {
 ff += *i*;

cout << "The incremented value of *ff* is " << *ff* << endl;

}

↓ very well defined

$i++$ ~~$+ i$~~ i

incremented value of i

$i++$ + i

undefined



GO CLASSES

Short Circuiting in C

```
#include <stdio.h>
int main()
{
    int a = 1, b = 1, c = 1;
    if (a == b || c++) {
        printf("%d", c);
    }
}
```

Not getting executed

↓

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

true → TRUE || c++

True ||

Anything

← True
=

we will not evaluate this

→ ALWAYS true

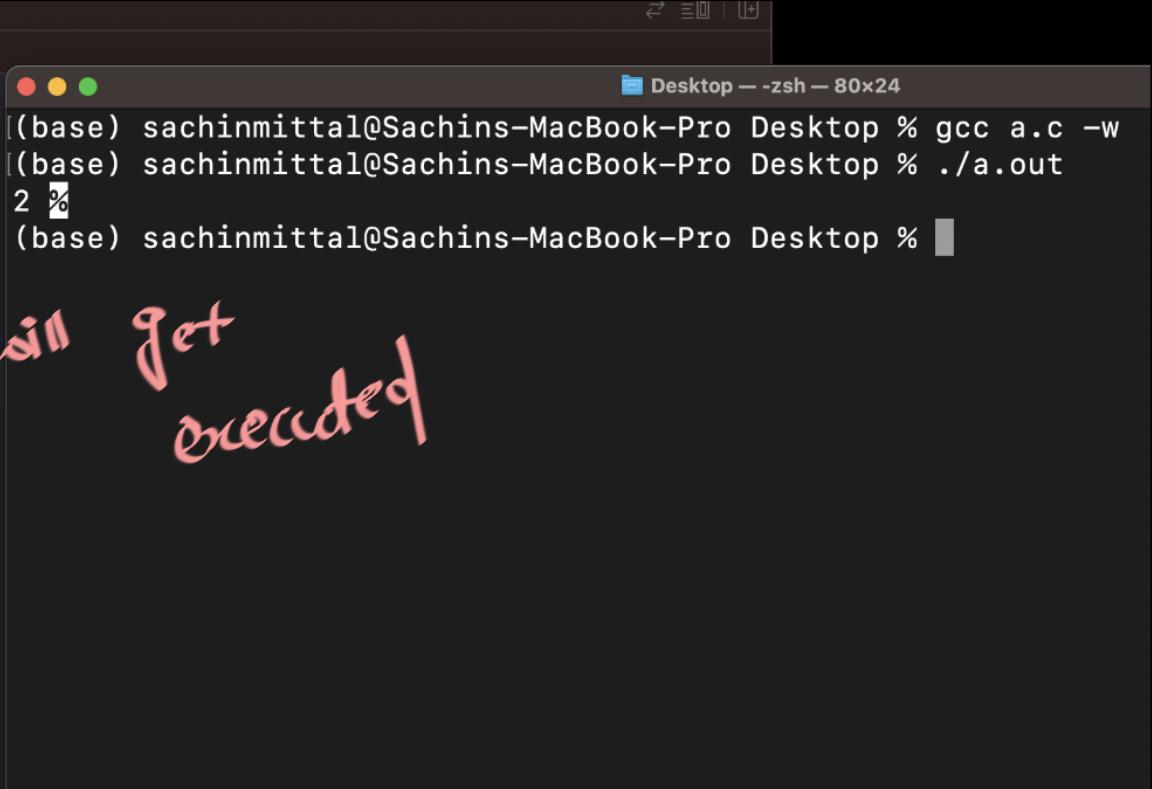
False ||

something

True

then final value is T

false
then final
is F



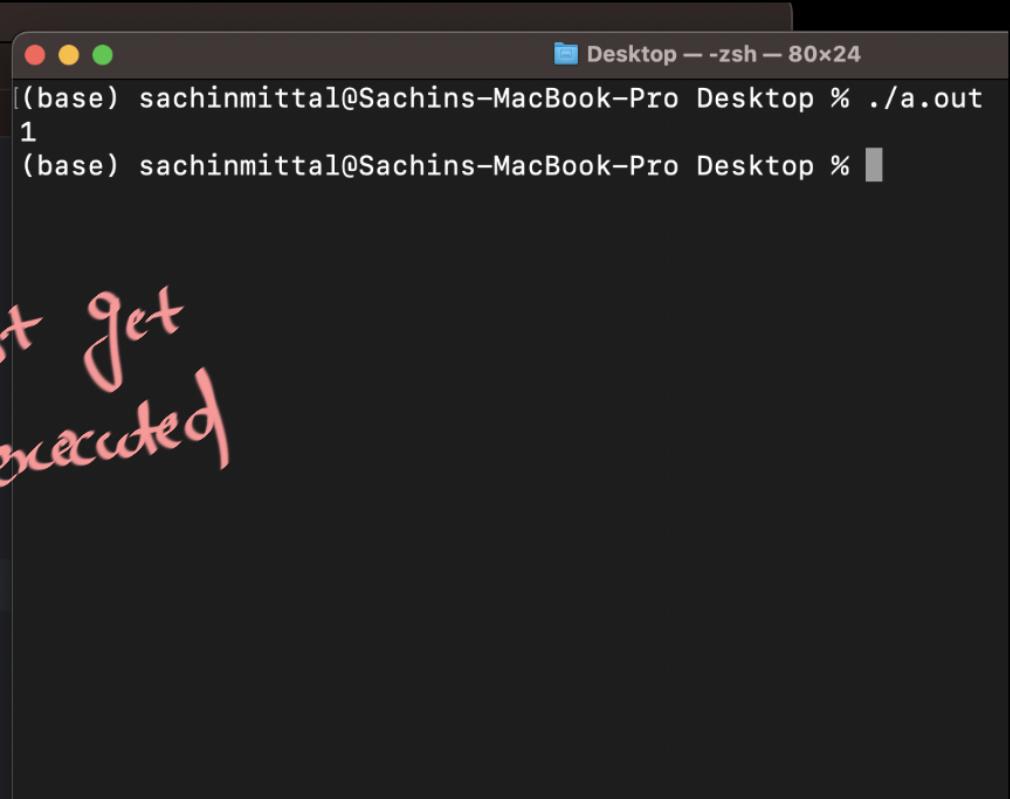
```
a.out:1: warning: control reaches end of non-void function [-Wreturn-type]
2
```

The image shows a terminal window titled "Desktop -- zsh -- 80x24" with the following text:

```
(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 %
```

Handwritten annotations in pink ink are present on the left side of the code editor:

- A red circle highlights the expression `c++` in the `if` condition.
- A red arrow points from the handwritten text "will get executed" down to the `c++` term in the code.



The image shows a terminal window titled "Desktop — zsh — 80x24" with the command "(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out" and the output "1". A handwritten note in pink ink says "will not get executed" with an arrow pointing to the circled part of the code.

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

will not get
executed



true || anything \Rightarrow true

GO
CLASSES

this will not get executed

Q. $\exp^1 \parallel \exp^2 \parallel \exp^3$

put the brackets correctly

Answer

$$\left(\exp^1 \parallel \exp^2 \right) \parallel \exp^3$$

Q. $\exp(1 + 4t + \exp^2) + 8t \exp^3$

put the brackets correctly

$$\left(\left(\exp(1 + 4t + \exp^2) \right) + 8t \exp^3 \right) \checkmark$$

Q: $\text{exp}^1 \neq \text{exp}^2 \parallel \text{exp}^3$

Put the brackets correctly

Answer

$$((\text{exp}^1 \neq \text{exp}^2) \parallel \text{exp}^3)$$

Q: $\text{expr}_1 \parallel \text{expr}_2 \neq \text{expr}_3$
Put the brackets correctly

Answer

$$\left(\text{expr}_1 \parallel \left(\text{expr}_2 \neq \text{expr}_3 \right) \right)$$

Q:

exp1 44 exp2 || exp3 88 exp4

put the brackets correctly

Answer:
$$\left(\text{exp1} \quad 44 \text{ exp2} \right) \parallel \left(\text{exp3} \quad 88 \text{ exp4} \right)$$

Q:

expr1 || expr2 & expr3 || expr4

Put the brackets correctly

Soln

$$\left(\text{expr1} \parallel \left(\text{expr2} \& \text{expr3} \right) \parallel \text{expr4} \right)$$



order of evaluation

expr || (expr & & expr)

CLASSES



C Programming

```
#include <stdio.h>
int main()
{
    int a = 1, b = 1, c = 1;
    if (a == b || c++) {
        printf("%d", c);
    }
}
```

TRUE ↗ *don't care*

↗ **CLASSES**



C Programming

```
#include <stdio.h>
int main()
{
    int a = 1, b = 1, c = 1;
    if (a != b || c++) {
        printf("%d", c);
    }
}
```

false *will get evaluated*

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



```
#include <stdio.h>
int main()
{
    int a = 1, b = 1, c = 1;
    if (a == b && c++)
        printf("%d", c);
}

```

TRUE TRUE ?? ++

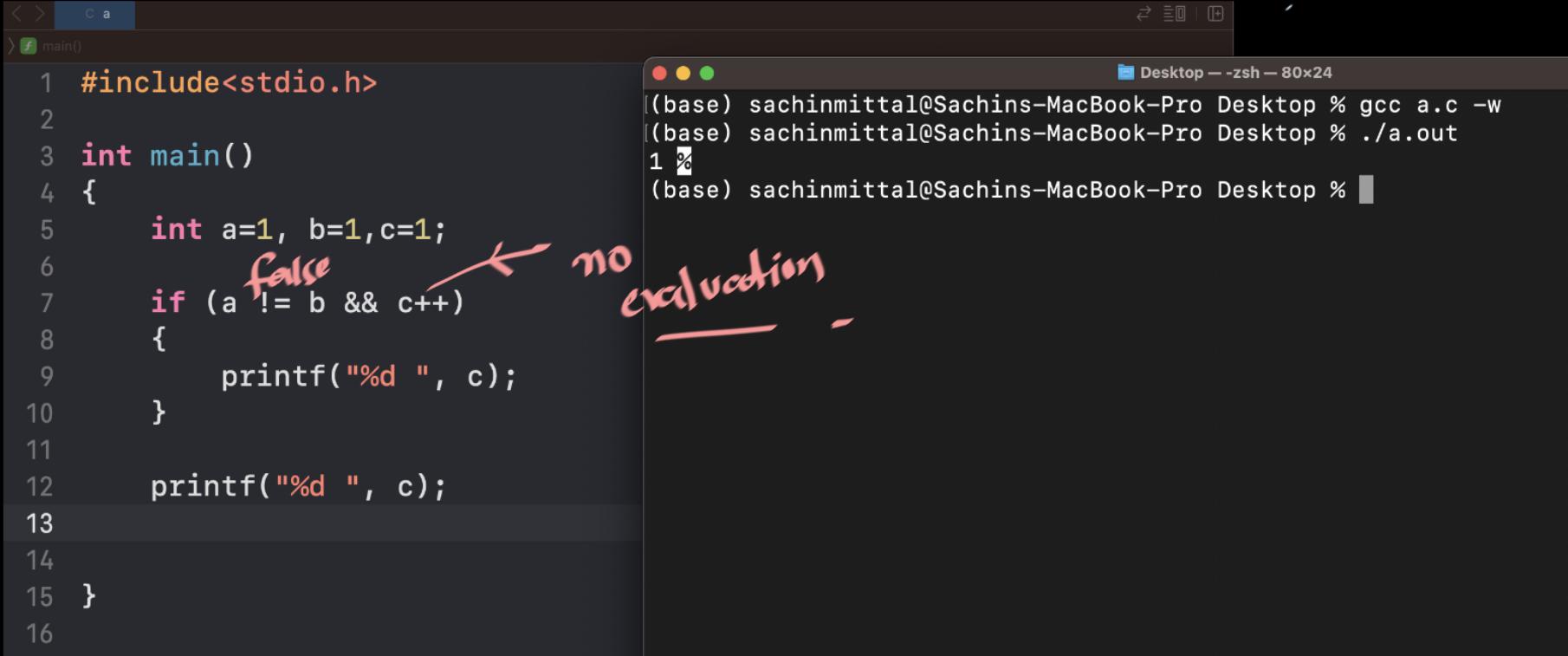


C Programming

```
#include <stdio.h>
int main()
{
    int a = 1, b = 1, c = 1;
    if (a != b && c++) {
        printf("%d", c);
    }
    printf("%d", c);
}
```

false for C++

false
==

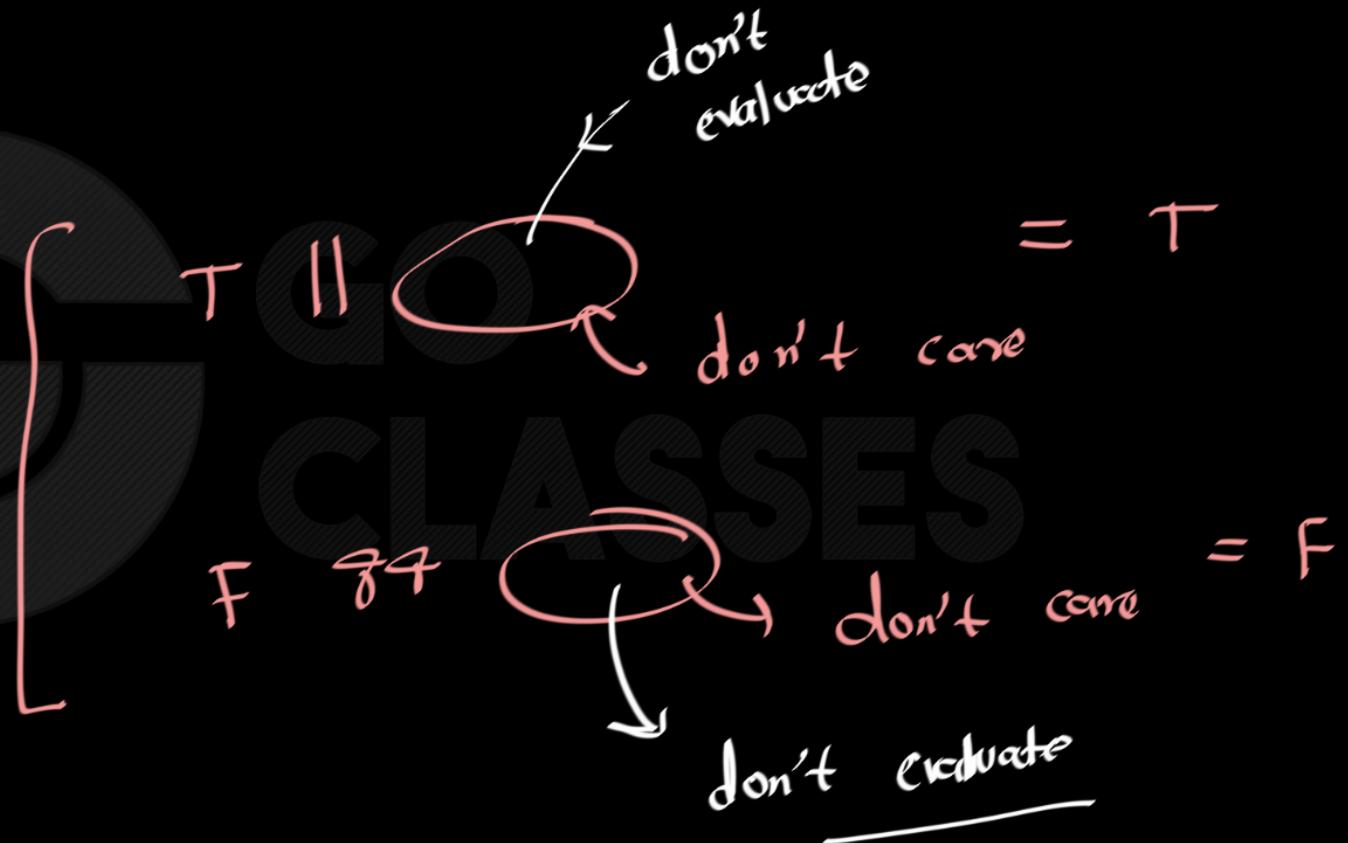


```
< > C a
> f main()
1 #include<stdio.h>
2
3 int main()
4 {
5     int a=1, b=1,c=1;
6     if (a != b && c++)
7     {
8         printf("%d ", c);
9     }
10    printf("%d ", c);
11
12
13
14
15 }
```

```
● ● ● Desktop -- zsh -- 80x24
(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 %
```

no evaluation

SHORT
CIRCUIT
BEHAVIOUR



Order of evaluation

int $\alpha = 1$; j ← will execute this first

int $b = \alpha + j$

Order of evaluation



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



C Programming

```
#include <stdio.h>
int main()
{
    int a = 1, b = 2, c = 1;
    if (a != b || c++) {
        printf("%d", c);
    }
}
```

TRUE

CLASSES



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

```
    int a = 1, b = 2, c = 1;
    TRUE                                ↗ No evaluation
    if (a != b || c++) {
        printf("%d", c);
    }
}
```



GATE 2021

```
#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;
}
```



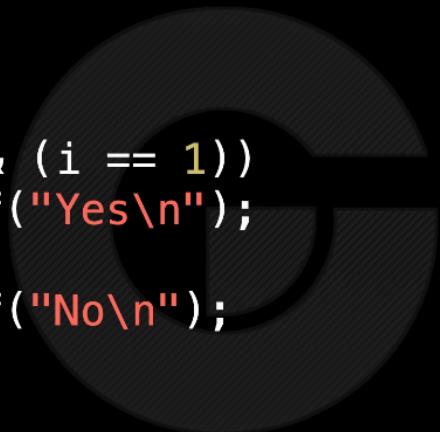
Which one of the following options is correct?

- 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 Programming

```
#include <stdio.h>
void main()
{
    int i = 1;
    if (i++ && (i == 1))
        printf("Yes\n");
    else
        printf("No\n");
}
```



GO
CLASSES
PLACEMENTS



```
#include <stdio.h>
void main()
{
    int i = 1;
    if (i++ && (i == 1))
        printf("Yes\n");
    else
        printf("No\n");
}
```

i is 2 here

if (i++ && (i == 1))

else

printf("Yes\n");

printf("No\n");

co

= No

π is a sequence point



C Programming

```
#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);
}
```

$$m = \left(\begin{matrix} i++ \\ O \end{matrix} \mid\mid \begin{matrix} j++ \\ I \end{matrix} \right) \mid\mid \begin{matrix} k++ \\ \uparrow \end{matrix};$$



C Programming

```
#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

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

No evaluation -

↓

1 2

bcoz expression is TRUE



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 -- 80x24". The command "gcc a.c -w" was run, followed by "./a.out". The output of the program is "1 1 2 2 %".

```
c a.c
> main()
1 #include<stdio.h>
2
3 int main()
4 {
5     int i = 0, j =1, k=2, m;
6
7     m = i++ || j++ || k++;
8
9
10    printf("%d %d %d %d ", m,i,j,k);
11
12
13 }
14
```

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
1 1 2 2 %
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```



C Programming

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

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

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

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



C Programming

```
#include <stdio.h>
void main()
{
    int x = 1, y = 0, z = 5;
    int a = x && y || z++;
    printf("%d", z);
}
```

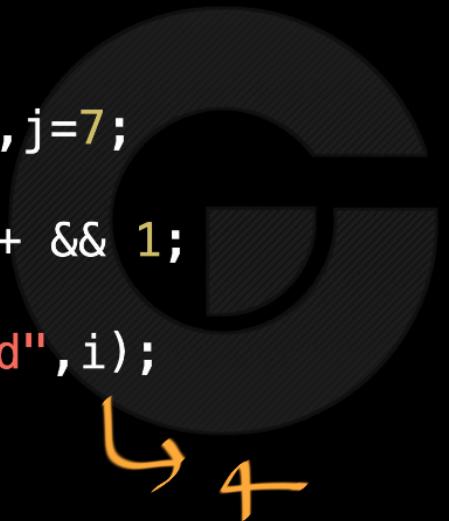
Handwritten annotations: A yellow arrow points from the value '1' to 'x'. A pink arrow points from '0' to 'y'. A pink arrow points from '5' to 'z'. Below the assignment statement, a pink arrow points from '5' to 'z'.

Handwritten annotations: A yellow circle highlights the value '1' next to 'x'. A pink circle highlights the value '0' next to 'y'. A pink arrow points from '0' to 'z++'.



C Programming

```
#include <stdio.h>
int main()
{
    int x, i=4, j=7;
    x=j || i++ && 1;
    printf("%d", i);
}
```



true
x=j || (i++ && 1);



C Programming

```
#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;
}
```



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



C Programming

```
#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;
}
```



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



C Programming

```
#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);
}
```



{ 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

Step 1 :

put the brackets

Step 2 :

go from left to right

Step 3 :

see if the short circuits are there
($T \parallel O$, $F \oplus Z$)