

Read 2 integers to perform bitwise operation
bitwise (&) AND, bitwise OR (|), bitwise NOT (~)
bitwise XOR (^)

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int num1, num2, result;
    clrscr();
    printf("Enter the first integer:");
    scanf("%d", &num1);
    printf("Enter the second integer:");
    scanf("%d", &num2);
    result = num1 & num2;
    printf("Bitwise & of %d and %d is: %d\n", num1, num2,
        result);
    result = num1 | num2;
    printf("Bitwise | of %d and %d is: %d\n", num1, num2, result);
    result = ~num1;
    printf("Bitwise ~ %d is: %d\n", num1, num2, result);
    result = num1 ^ num2;
    printf("Bitwise ^ of %d and %d is: %d\n", num1, num2, result);
    getch();
    return 0;
}
```

Output:

Enter the first integer: 4

Enter the second integer: 2

Bitwise & of 4 and 2 is: 0

Bitwise | of 4 and 2 is: 6

Bitwise ~ of 4 is: -5

Bitwise ^ of 4 and 2 is: 6

2) #include <stdio.h>

#include <conio.h>

int main()

{

int a, b;

clrscr();

printf("enter a value:");

scanf("%d", &a);

printf("enter b value:");

scanf("%d", &b);

printf("%d > %d is %d\n", a, b, a > b);

printf("%d < %d is %d\n", a, b, a < b);

printf("%d <= %d is %d\n", a, b, a <= b);

printf("%d >= %d is %d\n", a, b, a >= b);

printf("%d != %d is %d\n", a, b, a != b);

printf("%d == %d is %d\n", a, b, a == b);

Output:

enter a value: 4

enter b value: 2

4 > 2

4 < 2

$$4 > 2$$

$$4 \neq 2$$

$$4 == 2$$

3) mca on right shift and left shift and increment and decrement

i) what will be the value of x after the following code snippet executes

```
int x = 5;
x <<= 1;
```

a) 2 b) 5 c) 10 d) 25

Sol:- it is ~~xy~~ Left shift operation.

$$5 * 2^1 = 10$$

ii) #include <stdio.h>

#include <conio.h>

#include <math.h>

int main()

{ clrscr();

printf("2^5 using pow() function: %d of 1n",

pow(2, 5));

printf("2^5 using left shift: %d of 1n", (1 << 5));

getch();

return 0;

}

a) 36 b) 24 c) 16 d) 32

binary number of 1 is 1

binary number of 5 is 101

Left shift = $2^5 * 2^1$ pos

$$\begin{aligned}
 &= (1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3) \\
 &= 1 \ll 5 \text{ is } 1 * 2^5 \\
 &= 1 \times 32 \\
 &= 32
 \end{aligned}$$

(iii) #include <stdio.h>

#include <conio.h>

int main()

{

int N=3;

clrscr();

// left shift of 65 digits

printf ("3<<65 = %.d", (3<<65));

getch();

return 0;

}

a) 3<<650

b) 3<<750

c) 3<<65

d) 3<<100

Solu:- ~~3<<65~~ left shift

~~3~~ * 65

~~3~~

The code attempt to left shift integer 3 by 65 bits.

It is unsigned long long, shifting by 62 or more bits is undefined

So the output is 3<<650.

(iv) #include <stdio.h>

int main()

{

// left shift for negative value

printf ("2<<-5 = %.d\n", (2<<-5));

// Right shift

```
printf("2 >> -5 = %d", (2 >> -5));
```

```
return 0;
```

```
}
```

Q) What is the output of the program?

a) $2 \ll -5 = 0$

b) $2 \ll -5 = 0$

$2 \gg -5 = 65$

$2 \gg -5 = 64$

c) $2 \ll -5 = 64$

d) $2 \ll -5 = 0$

$2 \gg -5 = 0$

$2 \gg -5 = 32$

Sol: a) $2 \ll -5 = 0$

$2 \gg -5 = 0$

(V) What is the result of the expression $8 \ll 2$?

a) 2

b) 4

c) 16

d) 32

Sol: $8 \ll 2$

$= 8 * 2^2$

$= 8 * 4$

$= 32$

(VI) What is the result of the expression $100 \gg 3$?

a) 12

b) 25

c) 50

d) 100

Sol: $100 \gg 3$

$= 100 / 2^3$

$= 100 / 8$

$= 12$

(VII) Which of the following statements about the left shift operator (\ll) in C is true?

a) It divides the number by 2^n

b) It divides multiplies the number by 2^n

c) It performs a bitwise OR operation

d) It performs a bitwise XOR operation

As explained in Question 1, the left shift operator effectively multiplies the left operand by 2 raised to the power of the right operand.

```
vi) #include <stdio.h>
int main()
{
    int i = 2;
    i = i++ + i;
    printf("%d\n", i);
}
```

- a) = operator is not a sequence point
b) ++ operator may return value with or without side effects
c) it can be evaluated as $(i++) + i$ or $(i + i)$
d) = operator is a sequence point

Explanation: Because there is no sequence point between the two uses of i in $i++ + i$, the compiler has complete freedom to evaluate this in any way it sees fit. This could lead to different result on different compilers or even different optimization settings within the same compiler.

~~vi)~~ (ix) what will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int i = 2;
    int j = ++i + i;
    printf("%d\n", j);
}
```

- a) 6
b) 5
c) 4

d) compile

Explanation

$i++ + i$ becomes 3 the value

(X) what

inc

in

s

i

i

i

P

no

2

a) 0

Explana

$i++ + i$

increme

results

(xi) who

in

i

operator
2 raised

a) 6

b) 5

c) 4

d) compile time error

Explanation: i has an initial value of 2. In the expression $++i + i$, $++i$ increments i to 3, and the expression becomes $3 + 3$, which evaluates to 6. Hence, j contains the value 6.

(X) what will be the output of the following c code?

```
#include <stdio.h>
int main ()
{
    int i=0;
    int j = i++ + i;
    printf ("%d", j);
}
```

a) 0 b) 1 c) 2 d) compile time error.

Explanation: i has an initial value of 0. In the expression $i++ + i$, $i++$ uses the current value (0) and then increments i to 1, making the expression $0 + 1$, which results in 1. Hence, j contains the value 1.

ing c code?

(xi) what will be the output of the following c code?

```
#include <stdio.h>
int main()
{
    int a=10, b=10;
    if (a==5)
        b--;
    printf ("%d, %d", a, b--);
}
```

- a) $a=10, b=9$
- b) $a=10, b=8$
- c) $a=5, b=9$
- d) $a=5, b=8$

So $a=5$ assigns the literal 5 to variable a, and since $a=5$ is in a non-zero (true) expression, b- perform a post-decrement on b, changing its value from 10 to 9. Hence, a and b have values 5 and 9 respectively.

(Xii) For which of the following, " $p++$ ", code will compile?

- a) `#define p 3.14`
- b) `char *p = "A";`
- c) `float p = 3.14;`
- d) none of the mentioned

Ans:- The macro definition `#define p 3.14` substituted `p++` with `3.14`, and applying the `++` operator on a literal causes a compilation error.

(Xiii) what will be the output of the following code?

```
#include <stdio.h>

int main()
{
    int a=1, b=1, d=1;
    printf(" %d, %d, %d", ++a + ++a + a++);
}
```

- a) 15, 4, 5 b) 9, 6, 9 c) 9, 3, 5 d) undefined

Explanation: The provided code has several issues and undefined behavior due to multiple increments and decrements within a single expression, which can lead to different results depending on the compiler.

(Xiv) what will be the output of the following c code?

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

```
int main()
```

```
{
```

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

```
c = a++ + b
```

```
printf("%d", c, a, b);
```

```
}
```

a) a=1, b=1 b) a=2, b=1 c) a=1, b=2 d) a=2, b=2

the ++ operator has higher precedence than the + operator. So, in the expression $a++ + b$, i.e. $1++ + 1$, $a++$ (i.e. $1++$) increments a by 1, updating its value to 2. The value of b remains unchanged. Hence the output is 2, 1.

(Xv) #include <stdio.h>

```
int main()
```

```
{
```

```
int d, a=1, b=2;
```

```
d = a++ + ++b;
```

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

```
}
```

2) #include <stdio.h>

```
int main()
```

```
{
```

```
int d, a=1, b=2;
```

```
d = a++ + ++b;
```

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

```
}
```

What is the difference between the following 2 codes?

a) No difference as space doesn't make any difference, values of a, b, d are same in both the case.

b) Space does make a difference, values of a, b, d are different.

c) program 1 has syntax error, program 2 is not

d) program 2 has syntax error, program 1 is not

Sol:- $x+++y$ is not a valid operator in c. the tokeniser passes $+++$ as $x++$ (postfix increment) and $+y$. But $+++b$ is interpreted as $++b$, where the $++$ has no operand on its left. Hence, the compiler raises an error in program 2.