READ TWO INTGER VALUES AND PERFORM BITWISE OPERATIONS

INPUT:

```
#include<stdio.h>
int main() {
   int a,b;
   printf("enter a value");
   scanf("%d",&a);
   printf("enter b value");
   scanf("%d",&b);
   // Bitwise operations
   printf("\nBitwise operations:\n");
   printf("a & b = %d\n", a & b); // AND
   printf("a | b = %d\n", a | b); // OR
   printf("a ^ b = %d\n", a ^ b); // XOR
   printf("~a = %d\n",~a); // NOT A
   printf("~b = %d\n",~b); // NOT B
   // Perform shift operations
   printf("\nShift operations:\n");
   printf("a >> 1 = %d\n",a >> 1);
   printf("b >> 1 = %d\n",b >> 1);
   printf("a << 2 = %d\n",a << 2);
   printf("b << 2 = %d\n",b << 2);
   return 0;
}
```

OUTPUT:

```
enter a value6
enter b value3

Bitwise operations:
a & b = 2
a | b = 7
a ^ b = 5
~a = -7
~b = -4

Shift operations:
a >> 1 = 3
b >> 1 = 1
a << 2 = 24
b << 2 = 12


...Program finished with exit code 0
```

MCQs ON INCREMENT/DECREMENT AND SHIFT OPERATORS

1. What is the difference between the following 2 codes?

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

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

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
Answer: d
Explanation: x+++y is not a valid operator in C. The tokenizer parses +++ 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.

2. What will be the output of the following C code?

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

a)0, 2
b) 0, 1
c) 1, 2
d) Undefined
Answer: a
Explanation: In this program, the i is initialized with the value
i++ is a post-increment operation, so x is assigned the value of i (which is 0).
++i is a pre-increment operation, so i is incremented to 2 first, and then y is assigned the value of i (which is 2).

3. What will be the output of the following C code?

```c
#include <stdio.h>
int main()
{
   int c = 2 ^ 3;
   printf("%d\n", c);
}
```

a) 1
b) 8
c) 9
d) 7

Answer: a

Explanation:

- 0 XOR 0 is 0, and 1 XOR 1 is 0.
- 1 XOR 0 is 1, and 0 XOR 1 is 1.

4. What will be the output of the following C code?

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

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

Answer: b

Explanation: 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.

5. What will be the output of the following C code?

```c
#include <stdio.h>
int main()
{
    unsigned int a = 10;
    a = ~a;
    printf("%d\n", a);
}
```

a) -9
b) -10
c) -11
d) 10

Answer: c

Explanation: The ~ (tilde) operator is a bitwise operator that operates on an integer and returns its 1's complement.
Note: ~a is equivalent to -(a + 1).

6. What will be the output of the following C code?

```c
#include <stdio.h>
void main()
{
   int x = 97;
   int y = sizeof(x++);
   printf("X is %d", x);
}
```

a)X is 97
b) X is 98
c) X is 99
d) Run time error
Answer: a
Explanation: Since sizeof() is evaluated at compile time, the x++ does not affect the value of x during execution.

7. What will be the output of the following C code?

```c
#include <stdio.h>
int main()
{
   if (7 & 8)
   printf("Honesty");
      if (~7 & 0x000f) == 8)
         printf("is the best policy\n");
}
```

a)Honesty is the best policy
b) Honesty
c) is the best policy
d) No output
Answer: c
Explanation: The & in line 4 is a bitwise operator which returns the result of "7 & 8" as 0. So, line 5 is skipped.
In line 6, the ~7 is -8 and 0x000f is 15. Now the "-8 & 15" returns 8, then "8 == 8" is true i.e. 1. Hence, line 7 is executed and "is the best policy" is printed to console/screen.

8. What will be the output of the following C code?

```c
#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
Answer: c

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

9. Comment on the behaviour of the following C code?

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int i = 2;
5.      i = i++ + i;
6.      printf("%d\n", i);
7.  }
```

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
Answer: a
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 results on different compilers or even different optimization settings within the same compiler.

10. What will be the output of the following C code?

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int y = 0;
5.      if (1 |(y = 1))
6.          printf("y is %d\n", y);
7.      else
8.          printf("%d\n", y);
9.
10.  }
```

a) y is 1
b) 1
c) run time error
d) undefined
Answer: a
Explanation: The expression 1 | (y = 1) is evaluated as follows:
•First, y = 1 assigns the value 1 to the variable y.
•Then, y = 1 is replaced by 1 in the expression after the assignment.
•So, 1 | 1 (bitwise OR) evaluates to 1.
Thus, the if block is executed, and y is 1 is printed.

11. What will be the output of the following C code?

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

```
2.  void main()
3.  {
4.     int x = 4, y, z;
5.     y = –x;
6.     z = x–;
7.     printf("%d%d%d", x,  y, z);
8.  }
```

a)3 2 3
b) 2 3 3
c) 3 2 2
d) 2 3 4
Answer: b
Explanation: In the program, –x (pre-decrement) decreases the value of x before assignment, while x– (post-decrement) decreases the value of x after assignment.
Thus, the values of x, y, and z are 2, 3, and 3 respectively.

12. Comment on the output of the following C code.

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.     int i, n, a = 4;
5.     scanf("%d", &n);
6.     for (i = 0; i < n; i++)
7.        a = a * 2;
8.  }
```

a)Logical Shift left
b) Logical Shift Right
c) Arithmetic Shift right
d) Bitwise exclusive OR
Answer: a
Explanation: The given program performs the left logical/bitwise shift operation on variable a.

13. What will be the output of the following C code?

```
1.  #include <stdio.h>
2.  void main()
3.  {
4.     int a = 5, b = -7, c = 0, d;
5.     d = ++a && ++b || ++c;
6.     printf("\n%d %d %d %d", a, b, c, d);
7.  }
```

a) 6 -6 0 0
b) 6 -5 0 1
c) -6 -6 0 1
d) 6 -6 0 1
Answer: d
Explanation: In this code:

- ++a increments a to 6.
- ++b increments b to -6.
- The expression ++a && ++b evaluates to true (non-zero) because both ++a and ++b are non-zero.
- Since the left operand of the || operator is true, the right operand ++c is not evaluated due to short-circuiting.
- Therefore, d is assigned the value of ++a && ++b is true i.e. 1.

14. What will be the output of the following C code?

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int x = -2;
5.      x = x >> 1;
6.      printf("%d\n", x);
7.  }
```

a) 1
b) -1
c) $2^{31} - 1$ considering int to be 4 bytes
d) Either -1 or 1
Answer: b
Explanation: The program,
$(-2)_{10}$ => $(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110)_2$
On performing >> right shift on -2 by 1 bit makes all the bits to be 1.
-2 >> 1 => $(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111)_2$ => $(-1)_{10}$
Hence, -1 will be the output of the given program.

15. What will be the output of the following C code?

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      int x = -2;
5.      if (!0 == 1)
6.          printf("yes\n");
7.      else
8.          printf("no\n");
9.  }
```

a) yes
b) no
c) run time error
d) undefined
Answer: a
Explanation: The ! (NOT) operator inverts the logical boolean value of the operand. Therefore, !0 is 1, and 1 == 1 is true.
Hence, the if statement is executed, and yes is printed.