**Bitwise Operators**

**Left Shift Operators:**

* Refers to the process of shifting all the bits to the left in a number
* Example: 3 << 1 [Left shift 3 by 1]
* 3 can be represented as 0000 …011 in binary
* Left shift all the bits by 1
* Hence new representation will be 000….0110
* The new value of this binary representation is 6
* Left shift operators can also be defined as multiplying a number by 2

**Right Shift Operators:**

* Refers to the process of shifting all the bits to the right in a number.
* Example: 3 >> 1 [Right shift 3 by 1]
* 3 can be represented as 0000 …011 in binary
* Right shift all the bits by 1
* Hence new representation will be 000….0001
* The new value of this binary representation is 1
* Right shift operators can also be defined as dividing a number by 2

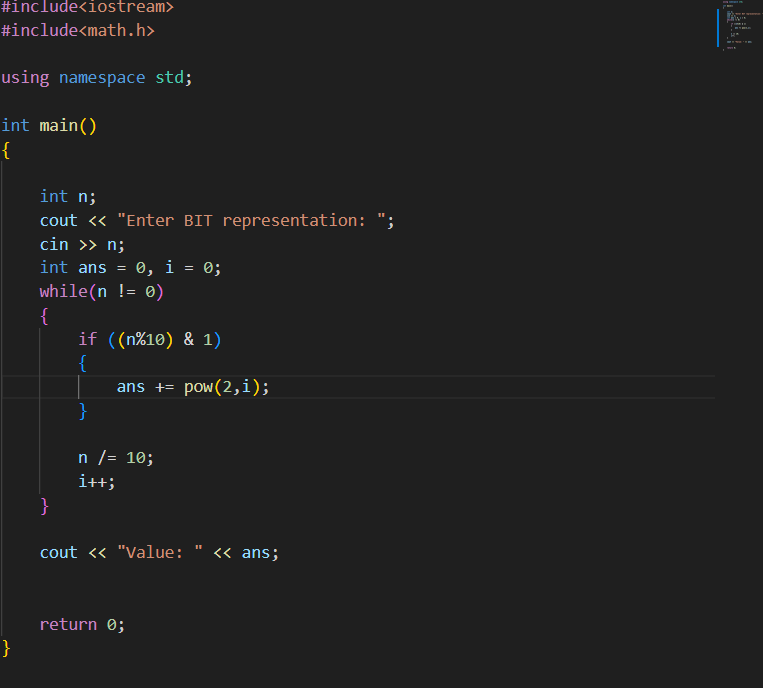
**Note:**

* **The Padding done in Bitwise operators in always 0.**
* **Bitwise Operators may produce unexpected results in very large numbers due to the sign changing between positive and negative.**
* **For Negative Numbers the padding is compiler dependent.**

**Questions:**

**Convert Binary Representation to Value:**

* **Code:**

****

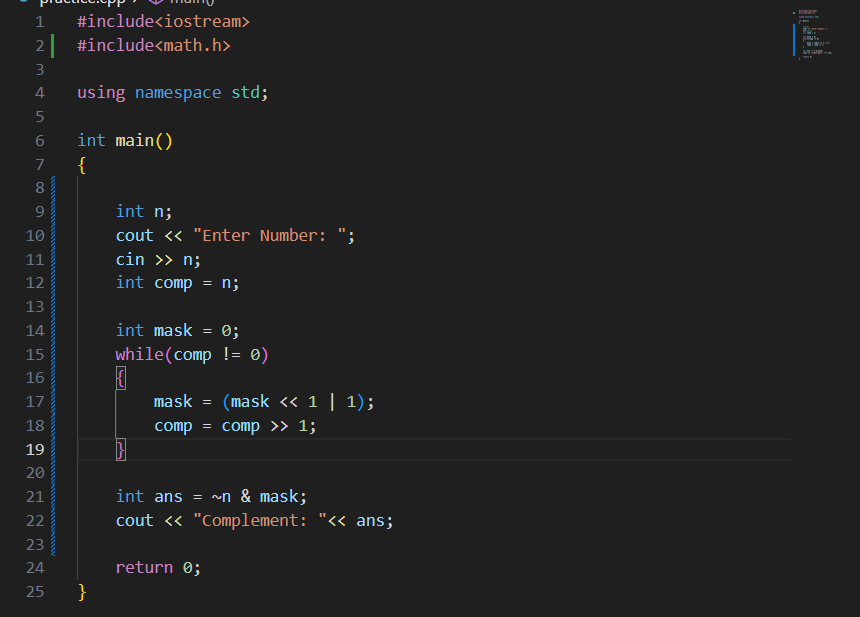
* **Output**

****

* **Explanation:**
* Initialize a variable to store the answer and the power of the place at which bits are present.
* Until the bit value reaches 0 start a loop to find the bits at which 1 is present.
* If the value of last digit is 1, we add the answer and power of 2 as the value increases by powers of 2 in binary operators.
* After that we divide the value by 10 to get the next last digit.
* Example: 00101The last digit is 1.
  + 1. The value of answer is 1
    2. The position of 1 is 0
    3. n % 10 returns 1 and we perform and operation to check if it is 1.
    4. Since it is 1 we then perform the operations ans = ans + 2^i
    5. Hence ans = (0 + 2^ 0) = (0 + 1) = 1
    6. We then divide n/10 which gives us 0010 and increment value of i
    7. Repeating this steps we get the value of binary representations.

**Print the complement of a number:**

* **Code:**

****

* **Output:**

****

* **Explanation:**
* Binary Representation of 7 is 0000….111
* Complement of 7 is 0000…000 which represents 0
* We know ~n converts all 0’s to 1’s and all 1’s to 0’s
* ~7 gives 1111…000
* **Purpose of the Mask:** The mask variable is used to ensure that the bitwise complement operation only affects the relevant bits of the input number n. Without the mask, the bitwise NOT operation would flip all bits, including leading zero bits, potentially leading to incorrect results for numbers with fewer bits.
* **How the Mask is Constructed:**
* The loop while(comp != 0) shifts mask left by one position and sets the least significant bit to 1 in each iteration. This continues until comp becomes zero.
* If n is a 4-bit number (e.g., 5, which is 0101 in binary), the mask will be constructed as 1111 in binary (which is 15 in decimal).
* **Final Operation:** The bitwise NOT of n is taken, and then the result is AND-ed with the mask to ensure that only the bits that correspond to the original number are considered, producing the correct complement.