1. Write a C program to input any number from user and check whether the Least Significant Bit (LSB) of the given number is set (1) or not (0). How to check whether the least significant bit of a number is set or unset using bitwise operator in C programming. C program to get the status of least significant bit of a number.

Example

Input

Input number: 11

Output

1.Write a C program to input any number from user and check whether the Least Significant Bit (LSB) of the given number is set (1) or not (0). How to check whether the least significant bit of a number is set or unset using bitwise operator in C programming. C program to get the status of least significant bit of a number.

Example

Input

Input number: 11

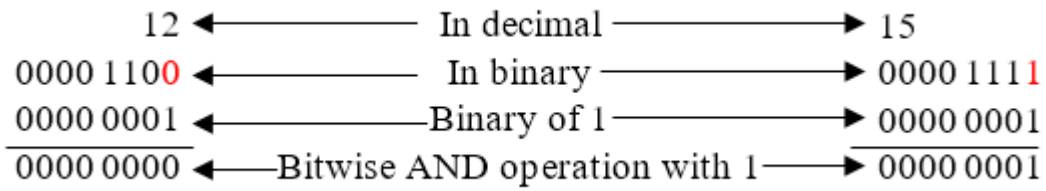
Output

Least Significant Bit of 11 is set (1).

Logic to check Least Significant Bit (LSB) of a number

We use Bitwise AND & operator to check status of any bit. Bitwise AND & operator evaluate each bit of the resultant value as 1, if corresponding bits of both operands are 1.

To check of a number we need to perform bitwise ANDing. The bitwise AND operation number & 1 will evaluate to 1 if LSB of number is set i.e. 1 otherwise evaluates to 0.



As you can see in above image 12 & 1 evaluate to 0. Since, LSB of 12 is 0. Whereas, 15 & 1 evaluate to 1 since LSB of 15 is 1.

Program to check Least Significant Bit (LSB) of a number

```
/**
 * C program to check Least Significant Bit (LSB) of a number using bitwise operator
 */

#include <stdio.h>
```

```
int main()
{
   int num;

   /* Input number from user */
   printf("Enter any number: ");
   scanf("%d", &num);

   /* If (num & 1) evaluates to 1 */
   if(num & 1)
      printf("LSB of %d is set (1).", num);
   else
      printf("LSB of %d is unset (0).", num);

   return 0;
}
```

Important note: The statement if(num & 1) is equivalent to if(num & 1 == 1).

Output

```
Enter any number: 11
LSB of 11 is set (1).
```

2. Write a C program to input any number from user and check whether Most Significant Bit (MSB) of given number is set (1) or not (0). How to check whether Most Significant Bit of any given number is set or not using bitwise operator in C programming. C program to get the status of the most significant bit of a number.

Example

Input

```
Input number: -1
```

Output

```
Most Significant Bit (MSB) of -1 is set (1).
```

Logic to check Most Significant Bit (MSB) of a number

We use bitwise AND & operator to check status of any bit. Bitwise AND operation evaluate each bit of resultant value as 1, if corresponding bit of operands is 1.

Step by step descriptive logic to check MSB of a number.

1. Input a number from user. Store it in some variable say *num*.
2. Find number of bits required to represent an integer in memory. Use sizeof() operator to find size of integer in bytes. Then multiply it by 8 to get number of bits required by integer. Store total bits in some variable say bits = sizeof(int) * 8;.

Read more - How to find size of a data type using sizeof() operator.

3. To get MSB of the number, move first bit of 1 to highest order. Left shift 1 bits - 1 times and store result in some variable say msb = 1 << (bits - 1).
4. If bitwise AND operation num & msb evaluate to 1 then MSB of *num* is set otherwise not.

Program to check Most Significant Bit (MSB) of a number

```
 * C program to check Most Significant Bit (MSB) of a number using bitwise operator
 */

#include <stdio.h>
#define BITS sizeof(int) * 8 // Total bits required to represent integer

int main()
{
   int num, msb;

   /* Input number from user */
   printf("Enter any number: ");
   scanf("%d", &num);

   /* Move first bit of 1 to highest order */
   msb = 1 << (BITS - 1);

   /* Perform bitwise AND with msb and num */
   if(num & msb)
      printf("MSB of %d is set (1).", num);
   else
      printf("MSB of %d is unset (0).", num);

   return 0;
}
```

The above condition if(num & msb) is equivalent to if((num & msb) == 1).

Important note: Most Significant Bit of positive number is always 0 (in 2s complement) and negative number is 1.

Output

```
Enter any number: -1
MSB of -1 is set (1).
```

3. Write a C program to input any number from user and check whether n^{th} bit of the given number is set (1) or not (0). How to check whether n^{th} bit of a given number is set or unset using bitwise operator in C programming. C program to get the status of n^{th} bit of a number.

Example

Input

```
Input number: 12
Input nth bit number: 2
```

Output

```
2 bit of 12 is set (1)
```

In previous two exercise, we learned to get Least Significant Bit (LSB) and Most Significant Bit (MSB) of a number. In this post we will learn to get nth bit of a number.

Logic to get nth bit of a number

Step by step descriptive logic to get nth bit of a number.

1. Input number from user. Store it in some variable say *num*.
2. Input the bit position from user. Store it in some variable say *n*.
3. To get the nth bit of *num* right shift *num*, *n* times. Then perform bitwise AND with 1 i.e. bitStatus = (num >> n) & 1;.

Program to get nth bit of a number

```
/**
 * C program to get the nth bit of a number
 */

#include <stdio.h>

int main()
{
   int num, n, bitStatus;

   /* Input number from user */
   printf("Enter any number: ");
   scanf("%d", &num);

   /* Input bit position you want to check */
   printf("Enter nth bit to check (0-31): ");
   scanf("%d", &n);

   /* Right shift num, n times and perform bitwise AND with 1 */
   bitStatus = (num >> n) & 1;

   printf("The %d bit is set to %d", n, bitStatus);

   return 0;
}
```

Always remember bit order starts from 0.

Output

```
Enter any number: 12
Enter nth bit to check (0-31): 2
The 2 bit is set to 1
```

4. Write a C program to input any number from user and set n^{th} bit of the given number as 1. How to set n^{th} bit of a given number using bitwise operator in C programming. Setting n^{th} bit of a given number using bitwise operator.

Example

Input

```
Input number: 12
Input nth bit to set: 0
```

Output

```
Number after setting nth bit: 13 in decimal
```

Logic to set nth bit of a number

We use bitwise OR | operator to set any bit of a number. Bitwise OR operator evaluate each bit of the resultant value to 1 if any of the operand corresponding bit is 1.

Step by step descriptive logic to set nth bit of a number.

1. Input number from user. Store it in some variable say *num*.
2. Input bit position you want to set. Store it in some variable say *n*.
3. To set a particular bit of number. Left shift 1 *n* times and perform bitwise OR operation with *num*. Which is newNum = (1 << n) | num;.

Program to set nth bit of a number

```
/**
 * C program to set the nth bit of a number
 */

#include <stdio.h>

int main()
{
   int num, n, newNum;

   /* Input number from user */
   printf("Enter any number: ");
   scanf("%d", &num);

   /* Input bit position you want to set */
   printf("Enter nth bit to set (0-31): ");
   scanf("%d", &n);

   /* Left shift 1, n times and perform bitwise OR with num */
   newNum = (1 << n) | num;

   printf("Bit set successfully.\n\n");
   printf("Number before setting %d bit: %d (in decimal)\n", n, num);
   printf("Number after setting %d bit: %d (in decimal)\n", n, newNum);

   return 0;
}
```

Output

```
Enter any number: 12
Enter nth bit to set (0-31): 0
```

Bit set successfully.

Number before setting 0 bit: 12 (in decimal)
Number after setting 0 bit: 13 (in decimal)

5. Write a C program to input any number from user and clear the n^{th} bit of the given number using bitwise operator. How to clear n^{th} bit of a given number using bitwise operator in C programming. How to unset (0) the value of n^{th} bit of a given number in C.

Example

Input

Input number: 13
Input nth bit to clear: 0

Output

Number after clearing nth bit: 12 (in decimal)

Also read -

- Program to get nth bit of a number.
- Program to set nth bit of a number.

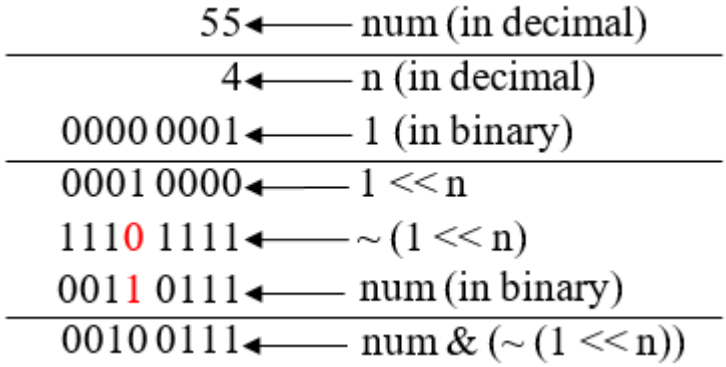
Logic to clear nth bit of a number

To clear n^{th} bit of a number we will use combination of bitwise left shift <<, bitwise complement ~ and bitwise AND & operator.

Below is the step by step descriptive logic to clear nth bit of a number.

1. Input number and nth bit position to clear from user. Store it in some variable say *num* and *n*.
2. Left shift 1, *n* times i.e. 1 << n.
3. Perform bitwise complement with the above result. So that the n^{th} bit becomes unset and rest of bit becomes set i.e. ~ (1 << n).
4. Finally, perform bitwise AND operation with the above result and *num*. The above three steps together can be written as num & (~ (1 << n));

Let us take an example to understand this.



Program to clear nth bit of a number

```
/**
 * C program to clear the nth bit of a number
 */

#include <stdio.h>

int main()
{
   int num, n, newNum;

   /* Input number from user */
   printf("Enter any number: ");
   scanf("%d", &num);

   /* Input bit number you want to clear */
   printf("Enter nth bit to clear (0-31): ");
   scanf("%d", &n);

   /*
    * Left shifts 1 to n times
    * Perform complement of above
    * finally perform bitwise AND with num and result of above
    */
   newNum = num & (~(1 << n));

   printf("Bit cleared successfully.\n\n");
   printf("Number before clearing %d bit: %d (in decimal)\n", n, num);
   printf("Number after clearing %d bit: %d (in decimal)\n", n, newNum);

   return 0;
}
```

Output

```
Enter any number: 13
Enter nth bit to clear (0-31): 0
Bit cleared successfully.

Number before clearing 0 bit: 13 (in decimal)
Number after clearing 0 bit: 12 (in decimal)
```

6. Write a C program to input any number from user and toggle or invert or flip n^{th} bit of the given number using bitwise operator. How to toggle n^{th} bit of a given number using bitwise operator in C programming. C program set n^{th} bit of a given number if it is unset otherwise unset if it is set.

Example

Input

```
Input number: 22
Input nth bit to toggle: 1
```

Output

```
After toggling nth bit: 20 (in decimal)
```

Required knowledge

Bitwise operators, Data types, Variables and Expressions, Basic input/output

Logic to toggle nth bit of a number

Toggling bit means setting a bit in its complement state. Means if bit is currently set then change it to unset and vice versa.

To toggle a bit we will use bitwise XOR `^` operator. Bitwise XOR operator evaluates to `1` if corresponding bit of both operands are different otherwise evaluates to `0`. We will use this ability of bitwise XOR operator to toggle a bit. For example - if Least Significant Bit of *num* is 1, then `num ^ 1` will make LSB of *num* to 0. And if LSB of *num* is 0, then `num ^ 1` will toggle LSB to 1.

Step by step descriptive logic to toggle nth bit of a number.

1. Input number and nth bit position to toggle from user. Store it in some variable say *num* and *n*.
2. Left shift 1 to *n* times, i.e. `1 << n`.
3. Perform bitwise XOR with *num* and result evaluated above i.e. `num ^ (1 << n);`.

Program to toggle or invert nth bit

```
/**
 * C program to toggle nth bit of a number
 */

#include <stdio.h>

int main()
{
    int num, n, newNum;

    /* Input number from user */
    printf("Enter any number: ");
    scanf("%d", &num);

    /* Input bit position you want to toggle */
    printf("Enter nth bit to toggle (0-31): ");
    scanf("%d", &n);

    /*
     * Left shifts 1, n times
     * then perform bitwise XOR with num
     */
    newNum = num ^ (1 << n);

    printf("Bit toggled successfully.\n\n");
    printf("Number before toggling %d bit: %d (in decimal)\n", n, num);
    printf("Number after toggling %d bit: %d (in decimal)\n", n, newNum);

    return 0;
}
```

Output

```
Enter any number: 22
Enter nth bit to toggle (0-31): 1
Bit toggled successfully.

Number before toggling 1 bit: 22 (in decimal)
Number after toggling 1 bit: 20 (in decimal)
```

7. Write a C program to input any number from user and find highest order set bit of given number using bitwise operator. How to find the highest order of the set bit of a given number using bitwise operator in C programming. Logic to get highest order set bit of a number in C programming.

Example

Input

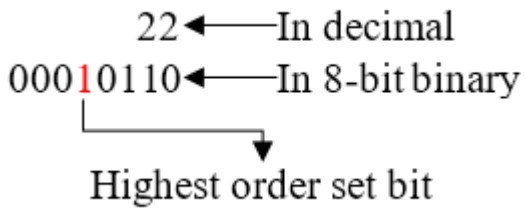
```
Input any number: 22
```

Output

```
Highest order set bit in 22 is 4.
```

Highest order set bit

Highest position of a set bit from left to right in a number is said to be highest order set bit of that number. Highest order set bit of any negative integers is 31 (for 32 bit signed integer). Since, highest order set bit of any negative number is its Most Significant Bit (MSB).



Logic to get highest order set bit of a number

Step by step descriptive logic to get highest order set bit of a number.

1. Input number from user. Store it in some variable say, *num*.
2. Find total bits required to store an integer in memory say, INT_SIZE = sizeof(int) * 8.

   Read more - How to find size of a data type using sizeof() operator.

3. Run a loop from 0 to *INT_SIZE*. The loop structure should look like for(i=0; i<INT_SIZE; i++).
4. Initialize a variable to store highest order, say order = -1;.
5. Inside the loop if i^{th} bit is set then update *order* to *i* i.e. order = i;.

Program to get highest order set bit of a number

```
/**
 * C program to find highest order set bit in a number
 */
```

```
#include <stdio.h>
#define INT_SIZE sizeof(int) * 8 /* Integer size in bits */

int main()
{
   int num, order = -1, i;

   /* Input number from user */
   printf("Enter any number: ");
   scanf("%d", &num);

   /* Iterate over each bit of integer */
   for(i=0; i<INT_SIZE; i++)
   {
      /* If current bit is set */
      if((num>>i) & 1)
         order = i;
   }

   if (order != -1)
      printf("Highest order set bit in %d is %d", num, order);
   else
      printf("0 has no set bits.");

   return 0;
}
```

Output

```
Enter any number: 22
Highest order set bit in 22 is 4
```

8. Write a C program to input any number from user and find lowest order set bit of given number using bitwise operator. How to find first set bit in a given number using bitwise operator in C programming. Logic to get first set bit of a number using C program.

Example

Input

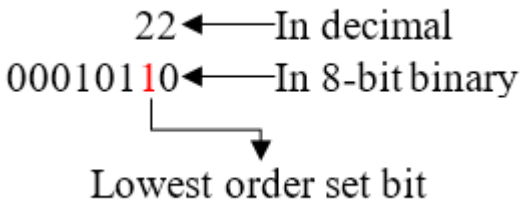
```
Input any number: 22
```

Output

```
First set bit: 1
```

Lowest order set bit

Lowest order or first set bit of any number is the first bit set starting from left to right. Lowest order set bit
of any odd number is 0 since first bit of any odd number is always set.



Logic to get lowest order set bit

Logic to get lowest order set bit is almost similar to highest order set bit of a number.

Step by step descriptive logic to get lowest order set bit.

1. Input number from user. Store it in some variable say, *num*.
2. Find total bits required to store an integer in memory say, INT_SIZE = sizeof(int) * 8.

   Read more - How to find size of a data type using sizeof() operator.

3. Run a loop from 0 to *INT_SIZE*. The loop structure should look like for(i=0; i<INT_SIZE; i++).
4. Initialize a variable to store lowest order set bit, say order = 0;.
5. Inside the loop if i^{th} bit is set then update *order* to *i* i.e. order = i; and terminate from loop.

Program to get lowest order set bit

```
/**
 * C program to get lowest order set bit in a number
 */

#include <stdio.h>
#define INT_SIZE sizeof(int) * 8 /* Integer size in bits */

int main()
{
   int num, order, i;

   /* Input number from user */
   printf("Enter any number: ");
   scanf("%d", &num);

   /* Initially set the order to max size of integer */
   order = INT_SIZE - 1;

   /* Iterate through each bit of integer */
   for(i=0; i<INT_SIZE; i++)
   {
      /* If current bit is set */
      if((num>>i) & 1)
      {
         order = i;

         /* Terminate the loop */
         break;
      }
   }
```

```
    printf("Lowest order set bit in %d is %d", num, order);

    return 0;
}
```

Output

```
Enter any number: 22
Lowest order set bit in 22 is 1
```

9. Write a C program to input any number from user and count number of trailing zeros in the given number using bitwise operator. How to find total number of trailing zeros in any given number using bitwise operator in C programming.

Example

Input

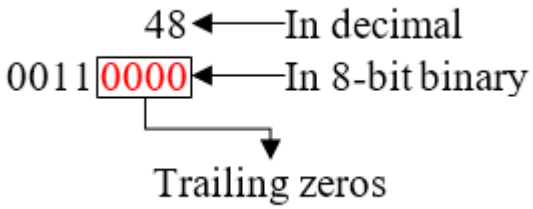
```
Input any number: 22
```

Output

```
Trailing zeros: 1
```

Logic to count trailing zeros in a binary number

Number of trailing zeros in a binary number is equal to first set bit position.



Step by step descriptive logic to count trailing zeros in a binary number.

1. Input number from user. Store it in some variable say, *num*.
2. Find total bits required to store an integer in memory say, INT_SIZE = sizeof(int) * 8.

   Read more - How to find size of a data type using sizeof() operator.

3. Initialize a variable to store trailing zeros count, say count = 0;.
4. Run a loop from 0 to *INT_SIZE*. The loop structure should look like for(i=0; i<INT_SIZE; i++).
5. Inside the loop if i^{th} bit is set then terminate from loop; otherwise increment *count* by 1.

Program to count trailing zeros in a binary number

```
/**
 * C program to count trailing zeros in a binary number using bitwise operator
 */

#include <stdio.h>
#define INT_SIZE sizeof(int) * 8 /* Bits required to represent an integer */
```

```
int main()
{
   int num, count, i;

   /* Input number from user */
   printf("Enter any number: ");
   scanf("%d", &num);

   count = 0;

   /* Iterate over each bit of the number */
   for(i=0; i<INT_SIZE; i++)
   {
      /* If set bit is found the terminate from loop*/
      if((num >> i ) & 1)
      {
         /* Terminate from loop */
         break;
      }

      /* Increment trailing zeros count */
      count++;
   }

   printf("Total number of trailing zeros in %d is %d.", num, count);

   return 0;
}
```

You can also short the above program using while loop.

Program to count trailing zeros in a binary number using while loop

```
/**
 * C program to count trailing zeros in a binary number using bitwise operator
 */

#include <stdio.h>

int main()
{
   int num, count=0;

   /* Input number from user */
   printf("Enter any number: ");
   scanf("%d", &num);

   while(!(num & 1))
   {
      count++;
      num >>= 1;
   }

   printf("Total number of trailing zeros = %d.", count);

   return 0;
}
```

Important note: The statement num >>= 1; is equivalent to num = num >> 1;.

Read more about - Shorthand assignment operators in C.

while(!(num & 1)) is equivalent to while((num & 1) == 0).

Learn more - Program to count leading zeros in a binary number

Output

```
Enter any number: 48
Total number of trailing zeros in 48 is 4.
```

10. Write a C program to input any number from user and find total number of leading zeros of the given number. How to find total leading zeros of a given number (in binary representation) using bitwise operator in C programming. Logic to count total leading zeros of a given number using C programming.

Example

Input

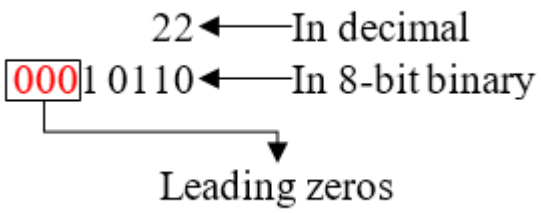
```
Input any number: 22
```

Output

```
Output leading zeros: 27 (using 4 byte signed integer)
```

Logic to count leading zeros in a binary number

Leading zeros in a binary number is equal to zeros preceding the highest order set bit of the number.



Step by step descriptive logic to count leading zeros in a binary number.

1. Input number from user. Store it in some variable say, *num*.
2. Find total bits required to store an integer in memory say, INT_SIZE = sizeof(int) * 8.

   Must know - How to find size of a data type using sizeof() operator.

3. Initialize a variable and set its to 1, say msb = 1 << (INT_SIZE - 1);.
4. Initialize a variable to store leading zeros count, say count = 0;.
5. Run a loop from 0 to *INT_SIZE*. The loop structure should look like for(i=0; i<INT_SIZE; i++).
6. Inside the loop, left shift *num*, *i* times and check its MSB is set or not. If its MSB is set i.e. if((num << i) & msb) then terminate the loop; otherwise increment *count* by 1.

Program to count leading zeros in a binary number

```
/**
 * C program to count leading zeros in a binary number using bitwise operator
 */

#include <stdio.h>
#define INT_SIZE sizeof(int) * 8

int main()
{
   int num, count, msb, i;

   /* Input number from user */
   printf("Enter any number: ");
   scanf("%d", &num);

   // Equivalent to
   // 10000000 00000000 00000000 00000000
   msb = 1 << (INT_SIZE - 1);

   count = 0;

   /* Iterate over each bit */
   for(i=0; i<INT_SIZE; i++)
   {
      /* If leading set bit is found */
      if((num << i) & msb)
      {
         /* Terminate the loop */
         break;
      }

      count++;
   }

   printf("Total number of leading zeros in %d is %d", num, count);

   return 0;
}
```

Below is another short and effective approach to solve the program.

Program to count leading zeros in a binary number using while loop

```
/**
 * C program to count leading zeros in a binary number using bitwise operator
 */

#include <stdio.h>
#include <limits.h> // Used for INT_MAX

int main()
{
   int num, count;

   /* Input number from user */
```

```
    printf("Enter any number: ");
    scanf("%d", &num);

    count = 0;
    while(!(num & (~INT_MAX)))
    {
        count++;
        num <<= 1;
    }

    printf("Total number of leading zeros = %d", count);

    return 0;
}
```

INT_MAX is a constant defined in limits.h header file. It is used to get maximum range of integer.

The statement num <<= 1; is a shorthand assignment operator equivalent to num = num << 1;

Output

```
Enter any number: 22
Total number of leading zeros in 22 is 27
```

11. Write a C program to input a number from user and flip all bits of the given number (in binary representation) using bitwise operator. How to flip all bits of a binary number using bitwise operator in C programming.

Example

Input

```
Input any number: 22
```

Output

```
Number after bits are flipped: -23 (in decimal)
```

Logic to flip all bits of a number

Flipping a bit means toggling or inverting the current bit status. If the current bit is set i.e. 1 than invert it to 0 and vice versa.

To flip all bits of a binary number you can run loop from 0 to size of the integer and flip individual bit at a time. However, C language has given bitwise complement ~ operator for the purpose.

Bitwise complement ~ evaluates complement of the operand bit. It evaluate to 1 if corresponding bit of the operand is 0 otherwise evaluate to 0.

Therefore to flip all bits of a number say *num*, you can use ~num.

Program to flip all bits of a number

```
/**
 * C program to count flip all bits of a binary number using bitwise operator
 */

#include <stdio.h>
```

```
int main()
{
   int num, flippedNumber;

   /* Input number from user */
   printf("Enter any number: ");
   scanf("%d", &num);

   flippedNumber = ~num;

   printf("Original number = %d (in decimal)\n", num);
   printf("Number after bits are flipped = %d (in decimal)", flippedNumber);

   return 0;
}
```

Output

```
Enter any number: 22
Original number = 22 (in decimal)
Number after bits are flipped = -23 (in decimal)
```

12. Write a C program to input a number from user and count total number of ones (1s) and zeros (0s) in the given number using bitwise operator. How to count zeros and ones in a binary number using bitwise operator in C programming.

Example

Input

```
Input any number: 22
```

Output

```
Output number of ones: 3
Output number of zeros: 29
```

Logic to count zeros and ones in a binary number

Step by step descriptive logic to count zeros and ones in a binary number.

1. Input a number from user. Store it in some variable say *num*.
2. Compute total bits required to store integer in memory i.e. INT_SIZE = sizeof(int) * 8.

   Must read - How to find size of a data type using sizeof() operator.

3. Initialize two variables to store zeros and ones count, say zeros = 0 and ones = 0.
4. Run a loop from 0 to INT_SIZE. The loop structure should look like for(i=0; i<INT_SIZE; i++).
5. Inside the loop check if Least Significant Bit of a number is set, then increment *ones* by 1 otherwise increment *zeros* by 1.
6. Right shift *num* 1 time i.e. perform num = num >> 1;.

Program to count zeros and ones in a binary number

```
/**
 * C program to count total of zeros and ones in a binary number using bitwise operator
```

```
 */

#include <stdio.h>
#define INT_SIZE sizeof(int) * 8 /* Total number of bits in integer */

int main()
{
   int num, zeros, ones, i;

   /* Input number from user */
   printf("Enter any number: ");
   scanf("%d", &num);

   zeros = 0;
   ones = 0;

   for(i=0; i<INT_SIZE; i++)
   {
      /* If LSB is set then increment ones otherwise zeros */
      if(num & 1)
         ones++;
      else
         zeros++;

      /* Right shift bits of num to one position */
      num >>= 1;
   }

   printf("Total zero bit is %d\n", zeros);
   printf("Total one bit is %d", ones);

   return 0;
}
```

In the above code if(num & 1) is equivalent to if((num & 1) == 1).
In the statement num >>= 1; I have used shorthand assignment operator which is equivalent to num = num >> 1;

Output

```
Enter any number: 22
Total zero bit is 29
Total one bit is 3
```

13. Write a C program to input a number and rotate bits of number using bitwise shift operators. How to rotate bits of a given number using bitwise operator in C programming. Logic to left or right rotate bits of a number using bitwise shift operator in C program.

Example

Input

```
Input number = -15
```

Number of rotations = 2

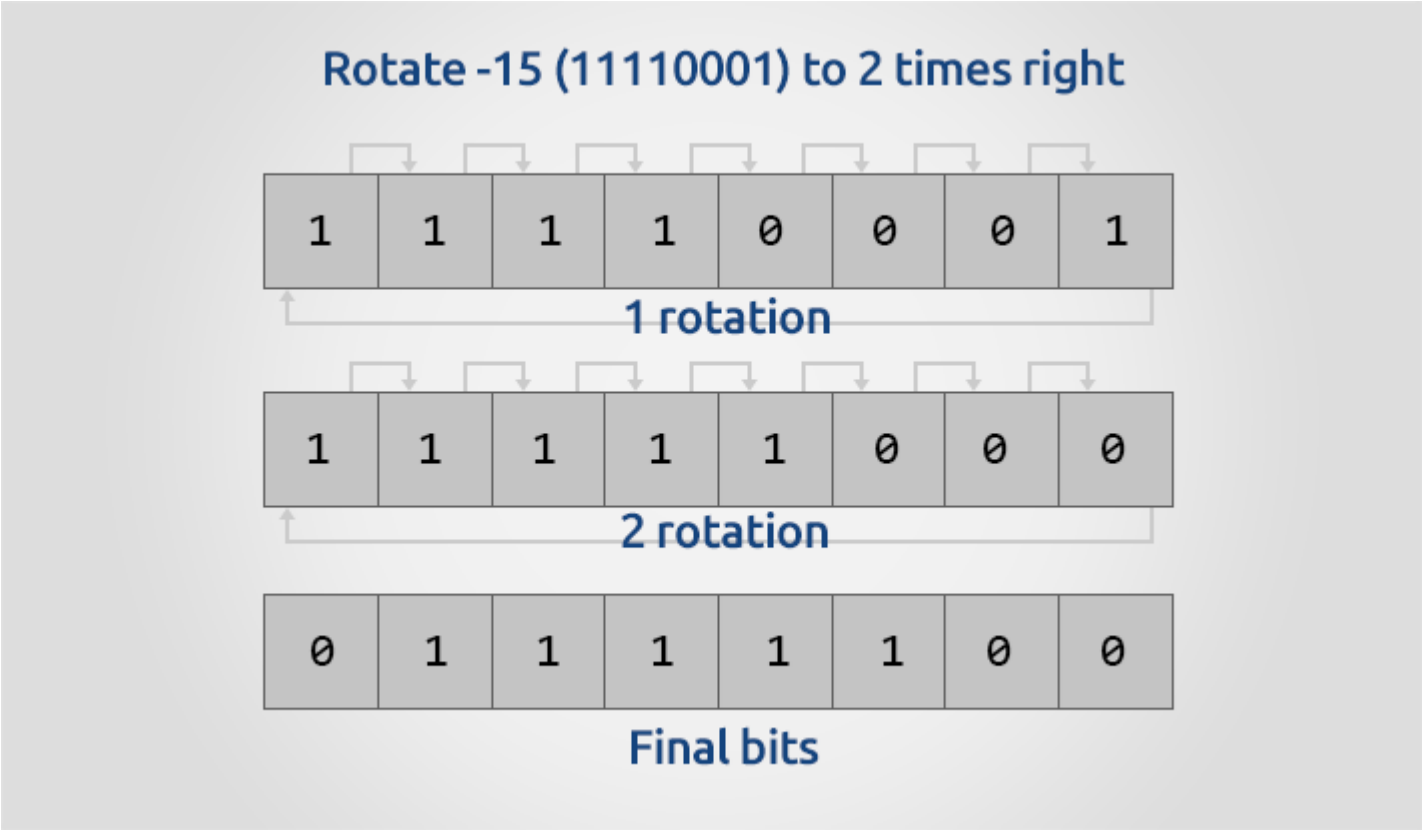
Output

-15 left rotated 2 times = -57
-15 right rotated 2 times = 2147483644

What is meant by rotating bits of a number

Rotating bits of a number to left, means shifting all bits to left and pushing the dropped Most Significant Bit to Least Significant Bit.

Rotating bits of a number to right, means shifting all bits to right and pushing the dropped Least Significant Bit to Most Significant Bit.



There is small difference in logic of left and right rotation of number. I have explained logic of both problems separately.

Logic to left rotate bits in a number

Left rotation of bits in C is supported using bitwise left shift operator <<. But left shift operator drops Most Significant Bit (MSB) on each shift. Which is what we don't want. Instead of dropping MSB on each rotation, Least Significant Bit (LSB) should get replaced as dropped MSB.

Step by step descriptive logic to left rotate bits of a number.

1. Input number to rotate and rotation amount from user. Store it in some variable say num and rotation.
2. Find most significant bit of the number that is about to drop and store it in some variable say DROPPED_MSB = (num >> INT_BITS) & 1.

3. Left Shift num one times and set its least significant bit to DROPPED_MSB i.e. perform num = (num << 1) | DROPPED_MSB.
4. Repeat step 2 and 3 for given number of rotations.

Logic to right rotate bits of a number

Right rotation of bits in C programming is supported using bitwise right shift operator >>. Similar to left shift, right shift operations also results in bit loss. On every shift operation the least significant bit is dropped.

What we need to do is, instead of dropping the least significant bit replace most significant bit with the dropped least significant bit.

Step by step descriptive logic to right rotate bits of a number.

1. Input number to rotate and rotation amount from user. Store it in some variable say num and rotation.
2. Find least significant bit of the number about to drop and store it in some variable say DROPPED_LSB = num & 1.
3. Right shift number one time i.e. perform num = num >> 1.
4. Clear most significant bit of number i.e. perform num = num & (~(1 << INT_BITS)).
5. Set the DROPPED_LSB as most significant bit of num. Perform num = num | (DROPPED_LSB << INT_BITS).
6. Repeat step 2 to 5 for given number of rotations.

Note: INT_BITS is total number of bits in integer - 1.

Program to rotate bits of a number

```
 * C program to rotate bits of a number.
 */


#include <stdio.h>

#define INT_SIZE sizeof(int)       // Size of int in bytes
#define INT_BITS INT_SIZE * 8 - 1   // Size of int in bits - 1


/* Function declarations */
int rotateLeft(int num, unsigned int rotation);
int rotateRight(int num, unsigned int rotation);


int main()
{
   int num;
   unsigned int rotation;

   /* Input number from user */
   printf("Enter a number: ");
   scanf("%d", &num);

   /* Input number of rotation */
```

```
    printf("Enter number of rotation: ");
    scanf("%u", &rotation);


    /* Print rotated number */
    printf("%d left rotated %u times = %d\n\n",  num, rotation, rotateLeft(num, rotation));
    printf("%d right rotated %u times = %d\n", num, rotation, rotateRight(num, rotation));


    return 0;
}



/**
 * Function to rotate bits of a number to left.
 *
 * @num        Number to rotate.
 * @rotation    Number of times to rotate left.
 */
int rotateLeft(int num, unsigned int rotation)
{
    int DROPPED_MSB;

    // The effective rotation
    rotation %= INT_BITS;


    // Loop till rotation becomes 0
    while(rotation--)
    {
        // Get MSB of num before it gets dropped
        DROPPED_MSB = (num >> INT_BITS) & 1;

        // Left rotate num by 1 and
        // Set its dropped MSB as new LSB
        num = (num << 1) | DROPPED_MSB;
    }

    return num;
}



/**
 * Function to rotate bits of a number to right.
 *
 * @num        Number to rotate.
 * @rotation    Number of times to rotate right.
 */
int rotateRight(int num, unsigned int rotation)
{
    int DROPPED_LSB;

    // The effective rotation
    rotation %= INT_BITS;


    // Loop till rotation becomes 0
```

```
    while(rotation--)
    {
        // Get LSB of num before it gets dropped
        DROPPED_LSB = num & 1;

        // Right shift num by 1 and
        // Clear its MSB
        num = (num >> 1) & (~(1 << INT_BITS));

        // Set its dropped LSB as new MSB
        num = num | (DROPPED_LSB << INT_BITS);
    }

    return num;
}
```

> **Note:** I have used %u format specifier for unsigned int. Learn about various format specifiers used in C.

To short the logic you can combine all bitwise operations as one. You can further decay above two functions as.

```
int rotateLeft(int num, unsigned int rotation)
{
    rotation %= INT_BITS;

    while(rotation--)
        num = (num << 1) | (1 & (num >> INT_BITS));

    return num;
}

int rotateRight(int num, unsigned int rotation)
{
    rotation %= INT_BITS;

    while(rotation--)
        num = ((num >> 1) & (~(1 << INT_BITS)) | ((num & 1) << INT_BITS));

    return num;
}
```

Output

```
Enter a number: -15
Enter number of rotation: 2

-15 left rotated 2 times = -57
-15 right rotated 2 times = 2147483644
```

14. Write a C program to input any decimal number from user and convert it to binary number system using bitwise operator. How to convert from decimal number system to binary number system using bitwise operator in C programming. Logic to convert decimal to binary using bitwise operator in C program.

Example

Input

Input any number: 22

Output

Binary number: 00000000000000000000000000010110

Logic to convert decimal to binary using bitwise operator

Step by step descriptive logic to convert decimal to binary number system.

1. Input a decimal number from user. Store it in some variable say *num*.
2. Declare an array of size required to store an integer in memory (i.e. 32 bits), say bin[INT_SIZE];.
3. Initialize another variable to store index, say index = INT_SIZE - 1;.
4. Run a loop from *INT_SIZE* to 0. Inside the loop assign Least Significant Bit to *bin[index]*. Perform bin[index] = num & 1;.
5. Decrement *index* by 1 and right shift *num* by 1.

Program to convert decimal to binary using bitwise operator

```
/**
 * C program to convert decimal to binary number system
 */

#include <stdio.h>
#define INT_SIZE sizeof(int) * 8 /* Size of int in bits */

int main()
{
   int num, index, i;
   int bin[INT_SIZE];

   /* Input number from user */
   printf("Enter any number: ");
   scanf("%d", &num);

   index = INT_SIZE - 1;

   while(index >= 0)
   {
      /* Store LSB of num to bin */
      bin[index] = num & 1;

      /* Decrement index */
      index--;

      /* Right Shift num by 1 */
      num >>= 1;
   }

   /* Print converted binary */
   printf("Converted binary: ");
   for(i=0; i<INT_SIZE; i++)
   {
      printf("%d", bin[i]);
   }
```

```
    return 0;
}
```

Output

```
Enter any number: 22
Converted binary : 00000000000000000000000000010110
```

15. Write a C program to input any two numbers from user and swap values of both numbers using bitwise operator. How to swap two number using bitwise operator in C programming. Logic to swap two numbers using bitwise operator in C programming.

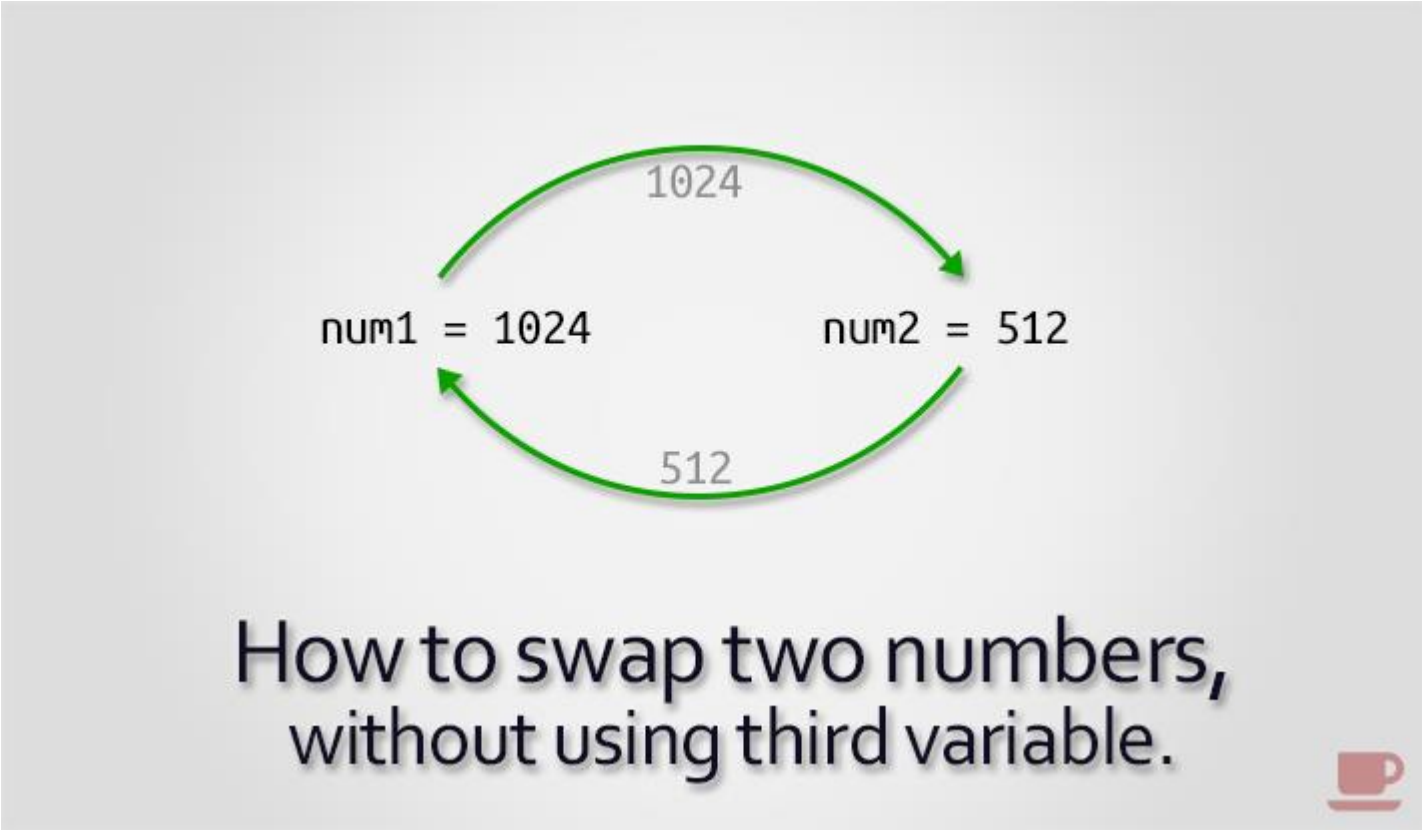
Example

Input

```
Input first number: 22
Input second number: 65
```

Output

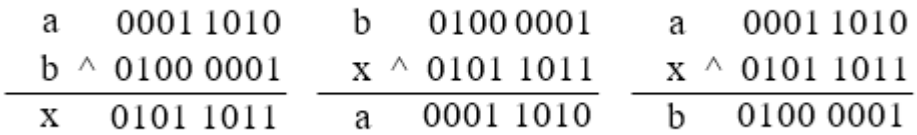
```
First number after swapping: 65
Second number after swapping: 22
```



Logic to swap two numbers using bitwise operator

There are tons of discussions going around the internet to swap two numbers without using temporary variable (third variable). We can use bitwise XOR ^ operator to swap to numbers. Bitwise XOR operator evaluates each bit of the result to 1 if corresponding bits of the operands are different otherwise evaluates 0.

Suppose two integer values *a* and *b*
Perform, x = a ^ b
Now x ^ b will evaluate to *a*
and x ^ a will evaluate to *b*.



Program to swap two numbers

```
/**
 * C program to swap two numbers using bitwise operator
 */

#include <stdio.h>

int main()
{
   int num1, num2;

   /* Input two numbers from user */
   printf("Enter any two numbers: ");
   scanf("%d%d", &num1, &num2);

   printf("Original value of num1 = %d\n", num1);
   printf("Original value of num2 = %d\n", num2);

   /* Swap two numbers */
   num1 ^= num2;
   num2 ^= num1;
   num1 ^= num2;

   printf("Num1 after swapping = %d\n", num1);
   printf("Num2 after swapping = %d\n", num2);

   return 0;
}
```

Output

```
Enter any two numbers: 22
65
Original value of num1 = 22
Original value of num2 = 65
Num1 after swapping = 65
Num2 after swapping = 22
```

16. Write a C program to input any number and check whether the given number is even or odd using bitwise operator. How to check whether a number is even or odd using bitwise operator in C programming. Logic to check even odd using bitwise operator in C programming.

Example

Input

Input number: 12

Output

12 is even

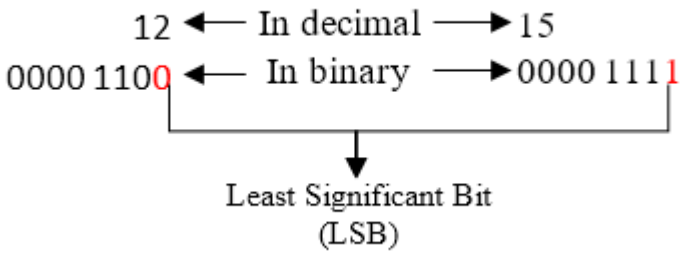
Must learn this program using other approaches.

Learn more -

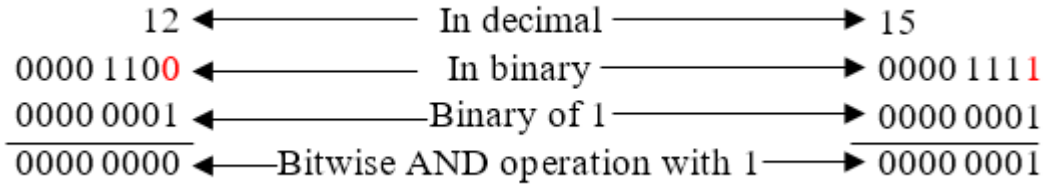
- Program to check even or odd using if else
- Program to check even or odd using switch case
- Program to check even or odd using conditional operator
- Program to check even or odd using functions

Logic to check even or odd using bitwise operator

Least Significant Bit of an odd number is always set (1). To check whether a number is even or odd we need to figure out if is set or not.



We use Bitwise AND & operator to check whether a bit is set or not. Similarly on performing num & 1 it return LSB of *num*. If LSB is 1 then the given number is odd otherwise even.



Program to check even or odd using bitwise operator

```
/**
 * C program to check even or odd number using bitwise operator
 */

#include <stdio.h>

int main()
{
   int num;
```

```
    /* Input number from user */
    printf("Enter any number: ");
    scanf("%d", &num);

    if(num & 1)
    {
        printf("%d is odd.", num);
    }
    else
    {
        printf("%d is even.", num);
    }

    return 0;
}
```

The statement if(num & 1) is equivalent to if((num & 1) == 1).

Note: You can also use conditional operator to short the program as done below.

Program to check even or odd using conditional and bitwise operator

```
/**
 * C program to check whether a number is even or odd using bitwise operator
 */

#include <stdio.h>

int main()
{
    int num;

    /* Input number from user */
    printf("Enter any number: ");
    scanf("%d", &num);

    (num & 1)
        ? printf("%d is odd.", num)
        : printf("%d is even.", num);

    return 0;
}
```

Output

```
Enter any number: 15
15 is odd.
```