1. Write a C program to determine if the least significant bit of a given integer is set (i.e., check if the number is odd).

```
#include <stdio.h>
int main() {
   int num;
   printf("Enter an integer: ");
   scanf("%d", &num);

   if (num & 1) {
      printf("The number is odd.\n");
   } else {
      printf("The number is even.\n");
   }

   return 0;
}
```

OUTPUT

```
Enter an integer: 5
The number is odd.
Enter an integer: 8
The number is even.
```

2. Create a C program that retrieves the value of the nth bit from a given integer.

```
#include <stdio.h>

int main() {
   int num, n;
   printf("Enter an integer: ");
```

```
    scanf("%d", &num);
    printf("Enter the bit position: ");
    scanf("%d", &n);

    int bit_value = (num >> n) & 1;
    printf("The %dth bit is: %d\n", n, bit_value);

    return 0;
}
```

OUTPUT

```
Enter an integer: 5
Enter the bit position: 2
The 2th bit is: 1
```

3. Develop a C program that sets the nth bit of a given integer to 1.

```
#include <stdio.h>

int main() {
    int num, n;
    printf("Enter an integer: ");
    scanf("%d", &num);
    printf("Enter the bit position to set: ");
    scanf("%d", &n);
    num = num | (1 << n);
    printf("New number: %d\n", num);

    return 0;
}
```

OUTPUT

```
Enter an integer: 8
Enter the bit position to set: 1
New number: 10
```

4. Write a C program that clears (sets to 0) the nth bit of a given integer.

```
#include <stdio.h>

int main() {
    int num, n;
    printf("Enter an integer: ");
    scanf("%d", &num);
    printf("Enter the bit position to clear: ");
    scanf("%d", &n);

    num = num & ~(1 << n);
    printf("New number: %d\n", num);

    return 0;
}
```

```
OUTPUT

Enter an integer: 15

Enter the bit position to clear: 3

New number: 7
```

5. Create a C program that toggles the nth bit of a given integer.

```
#include <stdio.h>


int main() {

   int num, n;

   printf("Enter an integer: ");

   scanf("%d", &num);

   printf("Enter the bit position to toggle: ");

   scanf("%d", &n);


   num = num ^ (1 << n);

   printf("New number: %d\n", num);


   return 0;

}
```

```
OUTPUT

Enter an integer: 10

Enter the bit position to toggle: 1

New number: 8
```

6. Write a C program that takes an integer input and multiplies it by 2^n using the left shift operator.

```
#include <stdio.h>

int main() {
   int num, n;
   printf("Enter a number: ");
   scanf("%d", &num);
   printf("Enter n: ");
   scanf("%d", &n);

   int result = num << n;
   printf("Result: %d\n", result);

   return 0;
}
```

OUTPUT

```
Enter a number: 3

Enter n: 2

Result: 12
```

7. Create a C program that counts how many times you can left shift a number before it overflows (exceeds the maximum value for an integer).

```
#include <stdio.h>


int main() {

   int num = 1, count = 0;


   while (num > 0) {
       num = num << 1;
       count++;
   }

   printf("Shifts before overflow: %d\n", count - 1);

   return 0;
}
```

OUTPUT

```
Shifts before overflow: 30
```

8. Write a C program that creates a bitmask with the first n bits set to 1 using the left shift operator.

```
#include <stdio.h>

int main() {
    int n;
    printf("Enter n: ");
    scanf("%d", &n);

    int bitmask = (1 << n) - 1;
    printf("Bitmask: %d\n", bitmask);

    return 0;
}
```

OUTPUT

```
Enter n: 5
```

```
Bitmask: 31
```

9. Develop a C program that reverses the bits of an integer using left shift and right shift operations.

```c
#include <stdio.h>

int main() {
  unsigned int num, reversed = 0;
  printf("Enter a number: ");
  scanf("%u", &num);

  for (int i = 0; i < 32; i++) {
    reversed = (reversed << 1) | (num & 1);
    num = num >> 1;
  }

  printf("Reversed: %u\n", reversed);

  return 0;
}
```

OUTPUT

```
Enter a number: 3
Reversed: 3221225472
```

10. Create a C program that performs a circular left shift on an integer.

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

```
int main() {
   unsigned int num, shifts;
   printf("Enter a number: ");
   scanf("%u", &num);
   printf("Enter shifts: ");
   scanf("%u", &shifts);

   unsigned int result = (num << shifts) | (num >> (32 - shifts));
   printf("Result: %u\n", result);

   return 0;
}
```

OUTPUT

```
Enter a number: 5
Enter shifts: 1
Result: 10
```

11. Write a C program that takes an integer input and divides it by 2^ n using the right shift operator.

```
#include <stdio.h>

int main() {
   int num, n;
   printf("Enter a number: ");
   scanf("%d", &num);
   printf("Enter n: ");
   scanf("%d", &n);
   int result = num >> n;
   printf("Result: %d\n", result);

   return 0;
}
```

OUTPUT

```
Enter a number: 16
Enter n: 2
Result: 4
```

12. Create a C program that counts how many times you can right shift a number before it becomes zero.

```
#include <stdio.h>

int main() {
   int num, count = 0;
   printf("Enter a number: ");
   scanf("%d", &num);

   while (num > 0) {
      num = num >> 1;
      count++;
   }

   printf("Right shifts before zero: %d\n", count);
```

```
    return 0;
}
```

OUTPUT

```
Enter a number: 18
Right shifts before zero: 5
```

13. Write a C program that extracts the last n bits from a given integer using the right shift operator.

```
#include <stdio.h>

int main() {
    int num, n;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("Enter n: ");
    scanf("%d", &n);

    int result = num & ((1 << n) - 1);
    printf("Last %d bits: %d\n", n, result);

    return 0;
}
```

OUTPUT

```
Enter a number: 29
Enter n: 3
Last 3 bits: 5
```

14. Develop a C program that uses the right shift operator to create a bitmask that checks if specific bits are set in an integer.

```
#include <stdio.h>


int main() {
    int num, n;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("Enter the bit position to check: ");
    scanf("%d", &n);


    if ((num >> n) & 1) {
        printf("Bit %d is set (1).\n", n);
    } else {
        printf("Bit %d is not set (0).\n", n);
    }


    return 0;
}
```

OUTPUT

```
Enter a number: 18
Enter the bit position to check: 1
Bit 1 is set (1).
```