STATIC:

```
#include<stdio.h>
void myfun(void);
int main(){
    myfun();
    myfun();
    myfun();
    myfun();

    return 0;

}
void myfun(){
    static int count=0;
    count=count+1;
    printf("The function is executed %d times \n",count);
}
```

EXTERN:

main.c

```
#include<stdio.h>
int mainPrivateData;
void Testfile_myfun();
int main(){
    mainPrivateData=100;
    printf("001mainPrivateData = %d\n",mainPrivateData);
    Testfile_myfun();
    printf("002mainPrivateData = %d",mainPrivateData);
    return 0;
}
```

Testfile.c

```
extern int mainPrivateData;


void Testfile_myfun(){
    mainPrivateData=500;
}
```

STATIC AND EXTERN:

main.c:

```
#include<stdio.h>
int mainPrivateData;
void Testfile_myfun();
int main(){
    Testfile_myfun();
    return 0;
}
static void change_clock(int system_clock){
    printf("System clock changed to %d \n",system_clock);
}
```

Testfile.c:

```
extern int mainPrivateData;
extern void change_clock(int);
void Testfile_myfun(){
    change_clock(500);
}
```

BITWISE OPERATORS

```
#include<stdio.h>
int main(){
    char A=40;
    char B=30;
    printf("The output after Bitwise OR(|) operation is %d \n",(A|B));
    printf("The output after Bitwise AND(&) operation is %d \n",(A&B));
    printf("The output after Bitwise XOR(^) operation is %d \n",(A^B));
    printf("The output after Bitwise NOT(~) operation is %d \n",(~A));
     return 0;
}
```

Output:

```
The output after Bitwise OR(|) operation is 62
The output after Bitwise AND(&) operation is 8
The output after Bitwise XOR(^) operation is 54
The output after Bitwise NOT(~) operation is -41
```

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("Number is odd");
    }
    else{
        printf("Number is even");
    }
    return 0;
}
```

Output:

```
Enter an integer: 7

Number is odd
```

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

```
#include<stdio.h>

int getNthbit(int num,int n){

   return (num>>n)&1;

}

int main(){

   int num,n;

   printf("Enter an integer: ");

   scanf("%d",&num);

   printf("Enter the bit position: ");

   scanf("%d",&n);

   int bitvalue=getNthbit(num,n);

   printf("The value of %d th  bit is %d",n,bitvalue);

   return 0;

}
```

Output:

```
Enter an integer: 29

Enter the bit position: 3

The value of 3 th  bit is 1
```

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

```
#include<stdio.h>

int setNthbit(int num,int n){

   int mask=1<<n;

   return num|mask;
}
int main(){
   int num,n;
   printf("Enter an integer: ");
   scanf("%d",&num);
   printf("Enter the bit position to set: ");
   scanf("%d",&n);
   int result=setNthbit(num,n);
   printf("Number after setting the nth bit is %d",result);
   return 0;
}
```

Output:

```
Enter an integer: 21
Enter the bit position to set: 3
Number after setting the nth bit is 29
```

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

```
#include<stdio.h>
int clearNthbit(int num,int n){
    int mask=~(1<<n);
    return num&mask;
}
int main(){
    int num,n;
    printf("Enter an integer: ");
    scanf("%d",&num);
    printf("Enter the bit position to clear: ");
    scanf("%d",&n);
    int result=clearNthbit(num,n);
    printf("Number after clearing the nth bit is %d",result);
    return 0;
}
```

Output:

```
Enter an integer: 126
Enter the bit position to clear: 1
Number after clearing the nth bit is 124
```

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

```
#include<stdio.h>
int toggleNthbit(int num,int n){
    int mask=1<<n;
    return num^mask;
}
int main(){
    int num,n;
    printf("Enter an integer: ");
    scanf("%d",&num);
    printf("Enter the bit position to toggle: ");
    scanf("%d",&n);
    int result=toggleNthbit(num,n);
    printf("Number after toggling the nth bit is %d",result);
    return 0;
}
```

Output:

```
Enter an integer: 72
Enter the bit position to toggle: 2
Number after toggling the nth bit is 76
```

Left shift

1. 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 the number: ");
    scanf("%d",&num);
    printf("Enter the value of n: ");
    scanf("%d",&n);
    int result = num<<n;
    printf("The result is %d",result);
    return 0;
}
```

Output:

```
Enter the number: 4
Enter the value of n: 2
The result is 16
```

2. 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).

3. 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 num;
    printf("Enter the number: ");
    scanf("%d",&num);
    int bitmask=(1<<num)-1;
    printf("The bitmask is 0x%X",bitmask);
    return 0;
}
```

Output:

```
Enter the number: 4
The bitmask is 0xF
```

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

```
#include <stdio.h>

unsigned int reverse_bits(unsigned int num) {
    unsigned int reversed_num = 0;
    int bit_count = 32;

    for (int i = 0; i < bit_count; i++) {
        reversed_num <<= 1;
        int bit = num & 1;
        reversed_num |= bit;
        num >>= 1;
    }
    return reversed_num;
}
    int main() {
    unsigned int num;
    printf("Enter a number: ");
    scanf("%u", &num);
    unsigned int reversed_num = reverse_bits(num);
    printf("Original number: %u\n", num);
    printf("Reversed number: %u\n", reversed_num);
    return 0;
}
```

Output:

```
Enter a number: 637219
Original number: 637219
Reversed number: 3298660352
```

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

```
#include <stdio.h>
unsigned int circular_left_shift(unsigned int num, int shift) {
    int size = sizeof(num) * 8;
    shift = shift % size;
    unsigned int shifted_num = (num << shift) | (num >> (size - shift));
    return shifted_num;
}
int main() {
    unsigned int num;
    int shift;
    printf("Enter an integer: ");
    scanf("%u", &num);
    printf("Enter the number of positions to shift: ");
    scanf("%d", &shift);
    unsigned int result = circular_left_shift(num, shift);
    printf("Original number: %u\n", num);
    printf("After circular left shift(s): %u\n", result);
    return 0;
}
```

Output:

```
Enter an integer: 5623
Enter the number of positions to shift: 3
```

```
Original number: 5623
After circular left shift(s): 44984
```

Right shift

1. 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 an integer: ");
  scanf("%d", &num);
  printf("Enter the number of positions to shift: ");
  scanf("%d", &n);
  int result = num >> n;
  printf("Result of dividing %d by 2^%d using right shift: %d\n", num, n, result);
  return 0;
}
```

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

```
#include <stdio.h>
int count(int num) {
  int shift_count = 0;
  while (num != 0) {
    num >>= 1;
    shift_count++;
  }
  return shift_count;
}
int main() {
  int num;
```

```
    printf("Enter an integer: ");
    scanf("%d", &num);
    int shifts = count(num);
    printf("The number can be right shifted %d times before it becomes zero.\n", shifts);
    return 0;
}
```

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

```
#include <stdio.h>
unsigned int extract_last_n_bits(unsigned int num, int n) {
    unsigned int mask = (1 << n) - 1;
    unsigned int result = num & mask;
    return result;
}
int main() {
    unsigned int num;
    int n;
    printf("Enter an integer: ");
    scanf("%u", &num);
    printf("Enter the number of last bits to extract: ");
    scanf("%d", &n);
    unsigned int last_n_bits = extract_last_n_bits(num, n);
    printf("The last %d bits of %u are: %u\n", n, num, last_n_bits);
    return 0;
}
```

4. 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 check(int num, int position[], int count) {
    int result = 1;
    for (int i = 0; i < count; i++) {
        int mask = 1 << position[i];
        if ((num & mask) == 0) {
            result = 0;
            break;
        }
    }
    return result;
}
int main() {
    int num;
    int count;
    printf("Enter an integer: ");
    scanf("%d", &num);
    printf("Enter the number of bits to check: ");
    scanf("%d", &count);
    int position[count];
    printf("Enter the bit positions to check (0-indexed from right):\n");
    for (int i = 0; i < count; i++) {
        scanf("%d", &position[i]);
    }
    if (check(num, position, count)) {
        printf("All specified bits are set to 1.\n");
    } else {
        printf("At least one of the specified bits is not set.\n");
    }
    return 0;}
```