Static storage class

```
#include <stdio.h>


void myFun(void);


int main(){
    myFun();
    myFun();
    myFun();
    myFun();
    //printf("002 The function is ececuted  %d times\n",count);
    return 0;
}


void myFun(){
    static int count = 0;
    count = count + 1;
    printf("001 The function is ececuted  %d times\n",count);
}
```

TestFile.c

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

Main.c

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

```
void TestFie_myFunc(void);
int mainPrivateData;//create a global variable
int main()
{
   mainPrivateData=100;
   printf("mainPrivateData= %d\n ",mainPrivateData);
   TestFile_myFunc();//calling fn
   printf("mainPrivateData= %d ",mainPrivateData);
   return 0;
}
```

Output

```
mainPrivateData= 100
 mainPrivateData= 500
```

Extern storage classes

Main.c

```
#include <stdio.h>
void TestFie_myFunc(void);
//int mainPrivateData;//create a global variable
int main()
{
  // mainPrivateData=100;
   //printf("mainPrivateData= %d\n ",mainPrivateData);
   TestFile_myFunc();//calling fn
  // printf("mainPrivateData= %d ",mainPrivateData);
   return 0;
}
```

```
static void change_clock(int system_clock){
   printf("System clock changed to=%d\n",system_clock);
}
```

TestFile.c

```
extern void change_clock(int);
extern int mainPrivateData;
void TestFile_myFunc(){
   change_clock(500);
   //mainPrivateData=500;
}
```

Output

System clock changed to=500

When we are using static storage class to a function

Main.c

```
#include <stdio.h>
void TestFie_myFunc(void);
//int mainPrivateData;//create a global variable
int main()
{
  // mainPrivateData=100;
   //printf("mainPrivateData= %d\n ",mainPrivateData);
   TestFile_myFunc();//calling fn
  // printf("mainPrivateData= %d ",mainPrivateData);
   return 0;
```

```
}

static void change_clock(int system_clock){
   printf("System clock changed to=%d\n",system_clock);
}
```

TestFile.c

```
extern void change_clock(int);
extern int mainPrivateData;
void TestFile_myFunc(){
   change_clock(500);
   //mainPrivateData=500;
}
```

Output

```
main.c: In function ‘main’:
main.c:16:5: warning: implicit declaration of function ‘TestFile_myFunc’; did you mea
n ‘TestFie_myFunc’? [-Wimplicit-function-declaration]
   16 |     TestFile_myFunc();//calling fn
      |     ^~~~~~~~~~~~~~~
      |     TestFie_myFunc
/usr/bin/ld: /tmp/ccmVmeBG.o: in function `TestFile_myFunc':
TestFile.c:(.text+0xe): undefined reference to `change_clock'
collect2: error: ld returned 1 exit status
```

here when we are using static in a fn in a file and that fn cant be accessed in other files.

Bitwise0iperator

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

Output

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

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).

   Program

```
#include <stdio.h>


int main() {
   int num;


   printf("Enter an integer: ");
   scanf("%d", &num);
```

```
    if (num & 1) {
        printf("The least significant bit is set. The number is odd.\n");
    } else {
        printf("The least significant bit is not set. The number is even.\n");
    }

    return 0;
}
```

Output

```
Enter an integer: 4
The least significant bit is not set. The number is even.
Enter an integer: 5
The least significant bit is set. The number is odd.
```

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

```
#include <stdio.h>

// Function to get the nth bit of a number
int getNthBit(int number, int n) {
    return (number >> n) & 1;
}

int main() {
```

```
    int number, n;

    // Get user input
    printf("Enter an integer: ");
    scanf("%d", &number);

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

    // Get the nth bit and print it
    printf("The %dth bit of %d is: %d\n", n, number, getNthBit(number, n));

    return 0;
}
```

```
Enter an integer: 2
Enter the bit position (n): 6
The 6th bit of 2 is: 0
```

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

```
#include <stdio.h>

// Function to set the nth bit of the number to 1
```

```
int setNthBit(int number, int n) {
    return number | (1 << n);  // OR the number with a mask where only nth bit is 1
}

int main() {
    int number, n;

    // Get user input
    printf("Enter an integer: ");
    scanf("%d", &number);

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

    // Set the nth bit to 1
    number = setNthBit(number, n);

    // Output the result
    printf("The new number after setting the %dth bit is: %d\n", n, number);

    return 0;
}
```

Output

Enter an integer: 2

Enter the bit position (n): 6

The new number after setting the 6th bit is: 66

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

```
#include <stdio.h>

void clearNthBit(int *num, int n) {
    // Create a mask with 1 at the nth bit position
    int mask = ~(1 << n);

    // Use bitwise AND to clear the nth bit
    *num &= mask;
}

int main() {
    int num, n;

    // Read input values
    printf("Enter an integer: ");
    scanf("%d", &num);
```

```
    printf("Enter the position of the bit to clear (0-based index): ");
    scanf("%d", &n);

    // Ensure the bit position is valid
    if (n < 0 || n >= sizeof(int) * 8) {
        printf("Invalid bit position\n");
        return 1;
    }

    // Clear the nth bit
    clearNthBit(&num, n);

    // Output the result
    printf("The number after clearing the %d-th bit is: %d\n", n, num);

    return 0;
}
```

Output

```
Enter an integer: 2
Enter the position of the bit to clear (0-based index): 6
The number after clearing the 6-th bit is: 2
```

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

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

```
// Function to toggle the nth bit of a number
int toggleNthBit(int num, int n) {
    // Create a mask where only the nth bit is 1
    int mask = 1 << n;

    // XOR the number with the mask to toggle the nth bit
    return num ^ mask;
}

int main() {
    int num, n;

    // Read the integer and the position of the bit to toggle
    printf("Enter an integer: ");
    scanf("%d", &num);

    printf("Enter the bit position to toggle (0-based index): ");
    scanf("%d", &n);

    // Toggle the nth bit and display the result
    int result = toggleNthBit(num, n);
    printf("The number after toggling the %d-th bit is: %d\n", n, result);

    return 0;
}
```

Output

```
Enter an integer: 2
Enter the bit position to toggle (0-based index): 6
```

The number after toggling the 6-th bit is: 66

Bitwise Shift operation

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 A,n;

    printf("Enter a number:");

    scanf("%d",&A);

    printf("Enter the value of n: ");

    scanf("%d", &n);

    int res=(A<<n);

   printf("The output is %d\n",res);

return 0;

}
```

```
Enter a number:3
Enter the value of n: 3
The output is 24
```

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).

```
#include <stdio.h>

#include <limits.h>

int main() {

   int num = 1;

   int shifts = 0;

   while (num <= INT_MAX / 2) {

      num = num << 1;

      shifts++;           }

   printf("The number of times you can left shift before overflow: %d\n", shifts);

   return 0;

}
```

output

The number of times you can left shift before overflow: 30

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 create_bitmask(int n) {
   return (1 << n) - 1}
int main() {
   int n;
   printf("Enter the value of n (number of bits to set): ");
   scanf("%d", &n);
   int bitmask = create_bitmask(n);
   printf("The bitmask with the first %d bits set to 1 is: %d (decimal)\n", n, bitmask);
   printf("In binary: ");
   for (int i = sizeof(bitmask) * 8 - 1; i >= 0; i--) {
```

```
            printf("%d", (bitmask >> i) & 1);
        }
    printf("\n");

        return 0;
    }
```

Output

```
Enter the value of n (number of bits to set): 2
The bitmask with the first 2 bits set to 1 is: 3 (decimal)
In binary: 00000000000000000000000000000011
```

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

```
 #include <stdio.h>

#include <limits.h>

unsigned int reverse_bits(unsigned int num) {

    unsigned int reversed = 0;

    int num_bits = sizeof(num) * 8;

    for (int i = 0; i < num_bits; i++) {

        reversed <<= 1;

            reversed |= (num & 1);

            num >>= 1;

    }



    return reversed;

}
```

```
int main() {
    unsigned int num;
    printf("Enter an integer: ");
    scanf("%u", &num);
    unsigned int reversed = reverse_bits(num);
    printf("Original number: %u\n", num);
    printf("Reversed number: %u\n", reversed);
    printf("Original number in binary: ");
    for (int i = sizeof(num) * 8 - 1; i >= 0; i--) {
        printf("%d", (num >> i) & 1);
    }
    printf("\n");

    printf("Reversed number in binary: ");
    for (int i = sizeof(reversed) * 8 - 1; i >= 0; i--) {
        printf("%d", (reversed >> i) & 1);
    }
    printf("\n");

    return 0;
}
```

Output

Enter an integer: 2

Original number: 2

Reversed number: 1073741824

Original number in binary: 00000000000000000000000000000010

Reversed number in binary: 01000000000000000000000000000000

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

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

```
    for (int i = sizeof(num) * 8 - 1; i >= 0; i--) {
        printf("%d", (num >> i) & 1);
    }
    printf("\n");

    printf("Shifted number in binary: ");
    for (int i = sizeof(result) * 8 - 1; i >= 0; i--) {
        printf("%d", (result >> i) & 1);
    }
    printf("\n");

    return 0;
}
```

Output

Enter an integer: 2

Enter the number of positions to shift: 5

Original number: 2

Number after circular left shift by 5 positions: 64

Original number in binary: 00000000000000000000000000000010

Shifted number in binary: 00000000000000000000000001000000

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 value of n: ");
    scanf("%d", &n);
    int result = num >> n;
    printf("%d divided by 2^%d is: %d\n", num, n, result);
    return 0;
}
```

Output

```
Enter an integer: 2
Enter the number of positions to shift: 5
Original number: 2
Number after circular left shift by 5 positions: 64
Original number in binary: 00000000000000000000000000000010
Shifted number in binary: 00000000000000000000000001000000
```

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_right_shifts(unsigned int num) {
    int shift_count = 0;
```

```
        while (num > 0) {
        num >>= 1;
        shift_count++;
    }
    return shift_count;
}
int main() {
    unsigned int num;
    printf("Enter an integer: ");
    scanf("%u", &num);
    int shifts = count_right_shifts(num);
    printf("The number can be right shifted %d times before it becomes zero.\n", shifts);
    return 0;
}
```

output

Enter an integer: 2

The number can be right shifted 2 times before it becomes zero

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

```
 #include <stdio.h>
#include <limits.h>
unsigned int extract_last_n_bits(unsigned int num, int n) {
```

```
    unsigned int mask = (1U << n) - 1;
    return num & mask;
}
int main() {
    unsigned int num;
    int n;
    printf("Enter an integer: ");
    scanf("%u", &num);
    printf("Enter the number of bits to extract from the end: ");
    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);
    printf("The last %d bits of %u in binary: ", n, num);
    for (int i = n - 1; i >= 0; i--) {
        printf("%d", (last_n_bits >> i) & 1);
    }
    printf("\n");
    return 0;
}
```

output

```
Enter an integer: 5
The number can be right shifted 3 times before it becomes zero.
Enter the number of bits to extract from the end: 2
```

The last 2 bits of 5 are: 1

The last 2 bits of 5 in binary: 01

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 is_bit_set(unsigned int num, int pos) {

    return (num >> pos) & 1;

}


int main() {

    unsigned int num;

    int pos;

    printf("Enter an integer: ");

    scanf("%u", &num);

    printf("Enter the position of the bit to check (0-indexed): ");

    scanf("%d", &pos);

    if (is_bit_set(num, pos)) {

        printf("The bit at position %d is set (1).\n", pos);

    } else {

        printf("The bit at position %d is not set (0).\n", pos);

    }
```

```
    return 0;
}
```

output

```
Enter an integer: 10
Enter the position of the bit to check (0-indexed): 6
The bit at position 6 is not set (0).
```