1.static

```
#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);
}
```

EXTERN

main.c

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

  mainPrivateData=100;

printf("001mainPrivateData=%d\n",mainPriv
ateData);
  TestFile_myFun();

printf("002mainPrivateData=%d",mainPrivat
eData);
  return 0;
```

TestFile.c

```
extern int mainPrivateData;

void TestFile_myFun(){
  mainPrivateData=500;

}
```

To prevent external access, make the variable **static-** *static int mainPrivateData*;

Extern function

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

  TestFile_myFun();
   return 0;
}

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

```
extern int mainPrivateData;
extern void change_clock(int);
void TestFile_myFun(){
  change_clock(500);

}
```

To prevent external access make the function **static**

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

Bitwise Operator

1. Test(&)
2. Clear(~ and &)
3. Set(|)
4. Toggle(^)

```
#include<stdio.h>
int main(){
  char a=40;
  char b=30;
  printf("after bitwise OR(|)is %d\n",(a|b));
  printf("after bitwise AND(&)is %d\n",(a&b));
  printf("after bitwise XOR(^)is %d\n",(a^b));
  printf("after bitwise NOT(~)is %d\n",(~b));

}
```

```
after bitwise OR(|)is 62
after bitwise AND(&)is 8
after bitwise XOR(^)is 54
after bitwise NOT(~)is -31
```

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 a=21;
  if((a&1)){
     printf("lsb is set");
  }
  else{
     printf("lsb not set");
  }

}
```

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

```
#include<stdio.h>
int main(){
  int num,bit;
  printf("enter rhe  number\n");
  scanf("%d",&num);
  printf("enter rhe bit number\n");
  scanf("%d",&bit);

  if((num>>bit)&1==1){
     printf("1");
  }
  else{
     printf("0");
  }

}
```

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

```
#include<stdio.h>

int main(){
  int num,bit;
  printf("enter rhe  number\n");
  scanf("%d",&num);
  printf("enter rhe bit number\n");
  scanf("%d",&bit);

  printf("after setting %d bit of %d is %d",bit,num,(1<<bit)|num);

}
```

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

```
#include<stdio.h>

int main(){
  int num,bit;
  printf("enter rhe  number\n");
  scanf("%d",&num);
  printf("enter rhe bit number\n");
  scanf("%d",&bit);

  printf("after setting %d bit of %d is %d",bit,num,num&~(1<<bit));

}
```

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

```
#include<stdio.h>

int main(){
  int num,bit;
  printf("enter rhe  number\n");
  scanf("%d",&num);
  printf("enter rhe bit number\n");
  scanf("%d",&bit);

  printf("after setting %d bit of %d is %d",bit,num,num^(1<<bit));

}
```

```
#include<stdio.h>

int main(){
  int num=0x1234;
  int mask1,mask2,out1,out2;

  mask1=((1<<4)|(1<<6));
  out1=(num|mask1);
  mask2=((1<<3)|(1<<9)|(1<<12));
  out2=(out1&~mask2);
  printf("output value is %x",out2);

}
```

---

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

```
#include<stdio.h>
#include<math.h>
int main(){
  int num,power;
  scanf("%d %d",&num,&power);

  printf("%d multiplied by 2^%d is %d",num,power,(num<<power));



}
```

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 <stdint.h>
int main() {
   uint8_t num, temp;
   int count = 1;
   printf("Enter a character: ");
   scanf("%hhd", &num);
   temp = num;
   while (num !=128) {
      num = num << 1;
      count++;
   }
   printf("Number of times '%hhd' can be left shifted before overflowing is %hhd\n", temp, count);
   return 0;
}
```

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 n,bitmask;
   printf("Enter n ");
   scanf("%d", &n);
   bitmask=(1<<n)-1;
   printf("bitmas with 1st %d numbers 1 is %d",n,bitmask);

   return 0;
}
```

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

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

1.Write a program to extract bits from 14th to 9th bits of a number.

```
#include <stdio.h>
for(int i=n;i>=0; i--)
{
if(num&(1<<i)) printf("1"); else printf("0");
}
{
int main()
{
unsigned int num = 0x1234;

int res=(num>>9) & 0x3F;

printf("\nThe number in is %d:",res);


}
```

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

```
#include<stdio.h>
#include<math.h>
int main(){
  int num,power;
  scanf("%d %d",&num,&power);

  printf("%d divided 2^%d is %d",num,power,(num>>power));


}
```

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

```
 #include<stdio.h>
#include<math.h>
int main(){
  int num,temp;
  scanf("%d",&num);
  int count=0;
  temp=num;
  while(num!=0){
     num=num>>1;
     count++;
  }

  printf("numer of times %d can be right shifted before overflowing is %d",temp,count);


}
```

2. 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, result;


   printf("Enter a number: ");
   scanf("%d", &num);
   printf("Enter the number of bits to extract: ");
   scanf("%d", &n);
```

```
    result = num & ((1 << n) - 1);


    printf("The last %d bits of %d are: %d\n", n, num, result);

    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 main() {
    int num, pos, result;


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

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


    result = (num >> pos) & 1;


    if (result == 1) {
        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;
}
```