第一题

```
//验证系统中右移操作是逻辑右移还是算术右移
#include <iostream>
using namespace std;
int main()
{
    char ch[8];
    int x  , y  ;// 输入 x > 0 ,  y < 0，便于比较
    cin >> x >> y ;
    cout << x << '\t' <<"的二进制数是" << itoa(x,ch,2) << endl ;

    int xx , yy ;
    xx = x >> 2;
    cout << x << '\t' << "右移 2 位后的二进制数是
" << itoa(xx,ch,2)<< endl;
    cout << y << '\t' << "的二进制数是" << itoa(y,ch,2)+16 << endl ;
    yy = y >> 2;
    cout << y << '\t' << "右移 2 位后的二进制数是
" << itoa(yy,ch,2)+16 << endl;
    return 0;
}
```

测试数据（输入一个正数和一个负数，便于比较）

7 -7

输出

```
7       的二进制数是111
7       右移2位后的二进制数是1
-7      的二进制数是1111111111111001
-7      右移2位后的二进制数是1111111111111110
```

结论：当正数右移时，高位补 0，当负数右移时，高位补 1

故一个整数右移时，高位补的时符号数，故该系统中右移操作时算术右移

第二题

```
//2 两个不同长度的带符号整型数进行逻辑运算时，若长度短的数是负数时，不同系统处
理方式可能不同，
//有的系统是高位补 0，有的系统是符号扩展，即高位补 1，
//试编写程序验证本系统在这种情况下是左端高位是补 0 还是补 1。
#include<iostream>
```

```cpp
using namespace std;

int main()
{
    char ch_x[8],ch_y[8];
    //cout << itoa(-1,ch,2) << endl;
    cout << "long long 的长度为" << sizeof(long long) << endl;
    cout << "------int 的长度为" << sizeof(int) << endl;
    cout << "short int 的长度为" << sizeof(short int) << endl;
    long long a = 1;//a = 0000 0000 0000 0000 0000 0000 0000 0000 0001
    int b = -1 ;//b = 1111 1111 1111 1111
    short int  c = -1;//c = 1111 1111

    int x = a | b ;
    cout << "a | b = " << x << "    的二进制
为 " << itoa(x,ch_x,2)  << endl;
    a = 1 , c = -1;//重置一下
    int y = a | c;
    cout << "a | c = " << y << "    的二进制
为 " << itoa(y,ch_y,2)  << endl;
    return 0;
}
```

输出

```
long long的长度为8
------int的长度为4
short int的长度为2
a | b = -1   的二进制为 11111111111111111111111111111111
a | c = -1   的二进制为 11111111111111111111111111111111
```

总结：如上；a = 0000 0000 0000 0000 0000 0000 0000 0001（longlong）

b = 1111 1111 1111 1111 1111 1111 1111 1111 （int）

a | b= 1111 1111 1111 1111 1111 1111 1111 1111

故可得，b 的高位是补 1 的，所以本系统左端高位是补 1

第三题

```cpp
//只用位运算将十进制数转化为十六进制数
//思路：利用位运算每次将输入数的二进制的后四位取出
#include <iostream>
#include <cstring>
using namespace std;
```

```
int main()
{
    char ch[]="0123456789ABCDEF";//定义索引表
    int n;
    cin >> n;
    unsigned x = (unsigned) n ;//转化为无符号型数字
    char st[8];//一般十六进制不会超过 8 位
    int i = 0;
    int c = ~ (~0 << 4);//c=000000001111
    while(x)
    {
        int d = x&c;//d 位 x 二进制数的后四位
        st[i++] = ch[d];//转化为十六进制传入
        x = x >> 4;//x 后移 4 位
    }
    for(int j = i-1 ; j >= 0; j -- )cout << st[j];//逆序存入的，逆序输出
    return 0;
}
```

测试数据(输入一个十进制数)

255

输出
```
255
FF
```

测试数据 2

-7

输出
```
-7
FFFFFFF9
```

均符合题意！

第四题

```
//只用位运算将十进制数转化为二进制数
#include <iostream>
using namespace std;

int main()
{
    char ch[]="01";
    char st[32];
    int n;
    cin >> n;
```

```
    unsigned x = (unsigned)n;

    int t = ~( ~ 0 << 1);//t=000000000001
    int i = 0;
    while(x)
    {
        int m = x&t;
        st[i++] = ch[m];
        x = x >> 1;
    }
    for(int j = i - 1 ; j >= 0 ; j --)cout << st[j];
    return 0;
}
```

测试数据(输入一个十进制数)

255

输出

```
255
11111111
```

-7

输出

```
-7
11111111111111111111111111111001
```

均符合题意!

第五题

```
//输入一个由 0、1 系列组成的 16 位二进制带符号数，转换为十六进制数和十进制数显示。
#include <iostream>
#include <cstring>
using namespace std;
char str[] = "0123456789ABCDEF";//定义索引表
char ch[20];
char ans[4];

char * transSixteen()//转化为十六进制
{
    int i = 0;
//求十六进制采用 每次求 4 的字节的十六进制
    for(int i = 0 ; i < 4 ; i ++)
    {
        int sum = 0;
```

```
        int t = 8;
        for(int j = i * 4 + i ; j < (i + 1) * 4 + i; j ++)//跳过空格
        {
            sum += (ch[j] - '0') * t;//累加
            t /= 2;
        }
        ans[i] = str[sum];//传入
    }
    return ans;
}

int transTen()//转化为十进制
{
    int sum = 0;
    int t = 1;
    for(int i = strlen(ch) - 1 ; i >= 0 ; i --)
    {
        if(ch[i] == ' ')continue;
        sum += (ch[i] - '0') * t ;//数学法依次累加
        t *= 2;
    }

    //查看符号位，若为1，则表示负数，则需要减去八位二进制最大的十进制数
    if(ch[0] == '1'){
        int t = 1;
        for(int i = 0 ; i < 16 ; i ++ )t *= 2;
        sum -= t;
    }

    return sum;
}

int main()
{

    int i = 0;
    char c = getchar();
    while(c != '\n')
    {
        ch[i++] = c;
        c = getchar();
    }

    puts(transSixteen());//输出十六进制
```

```
    int Ten = transTen();//输出十进制
    cout << Ten << endl;

    return 0;

}
```

测试数据 1 输入一个 16 字节的二进制数，每四个字节隔一个空格)

0000 0000 0011 1001

输出
```
0000 0000 0011 1001
0039
57
```

测试数据 2

1111 1111 1100 0111

输出
```
1111 1111 1100 0111
FFC7
-57
```

均符合题意

第六题

```
#include <iostream>
using namespace std;

typedef struct Data{
    int a : 3 ;
    int b : 5 ;
    int c : 6 ;
    int d : 9 ;
}Data;

int main()
{
    Data num;
    int x,y,z,t;
    scanf("%d%d%d%d",&x,&y,&z,&t);
    num = {x,y,z,t};

    char ch[16];//用于输出二进制
    printf("%d %x %s\n",num.a,num.a,itoa(num.a,ch,2));
```

```
    printf("%d %x %s\n",num.b,num.b,itoa(num.b,ch,2));
    printf("%d %x %s\n",num.c,num.c,itoa(num.c,ch,2));
    printf("%d %x %s\n",num.d,num.d,itoa(num.d,ch,2));
    return 0;
}
```

测试数据，输入 4 个十进制数字

3 16 63 250

输出

```
3 16 63 250
3 3 11
-16 fffffff0 11111111111111111111111111110000
-1 ffffffff 11111111111111111111111111111111
250 fa 11111010
```

总结：具有位段的成员是按照位域内最高位自动补全高位的

第七题

```
//按 4 字节浮点数格式用 0 和 1 组成的字符串从键盘输入，转换为十进制数显示
#include<iostream>
#include<cstring>
#include<cmath>
using namespace std;

int main()
{
    char st[40];//输入的字符串
    char c;
    c = getchar();

    int j = 0 ;
    while(c != '\n')//输入二进制字符串
    {
        st[j++] = c;
        c = getchar();
    }

    double res ;//res 是最后的结果
    int index = 0 ;//表示指数

    int t = 128;
    for(int i = 2 ; i < 11 ; i ++ )
    {
        if(st[i] == ' ')continue;//跳过空格
        index += ( st[i]- '0' ) * t ;
```

```
        t /= 2 ;
    }

    index -= 127;//减去 0111 1111 得到最终的指数

    res = pow(2 , index);//求 res
    for(int i = 11 ; i < strlen(st) ; i ++ )
    {
        if(st[i] == ' ')continue;//跳过空格
        double x = pow(2 , --index);
        res += (st[i] - '0') * x ;//求得当前位的值，并累加至 res 中
    }

    if(st[0] == '1')res = -res; //判断符号位
    printf("%g\n",res);//舍去末尾的 0 输出
    return 0;
}
```

测试数据 1（输入二进制浮点数，有空格）

0 1000 0100 1100 1100 1000 1111 0101 110

```
0 1000 0100 1100 1100 1000 1111 0101 110
57.57
```

输出

测试数据 2

1 1000 0100 1100 1100 1000 1111 0101 110

```
1 1000 0100 1100 1100 1000 1111 0101 110
-57.57
```

符合题意！

按照浮点数运算法则计算即可