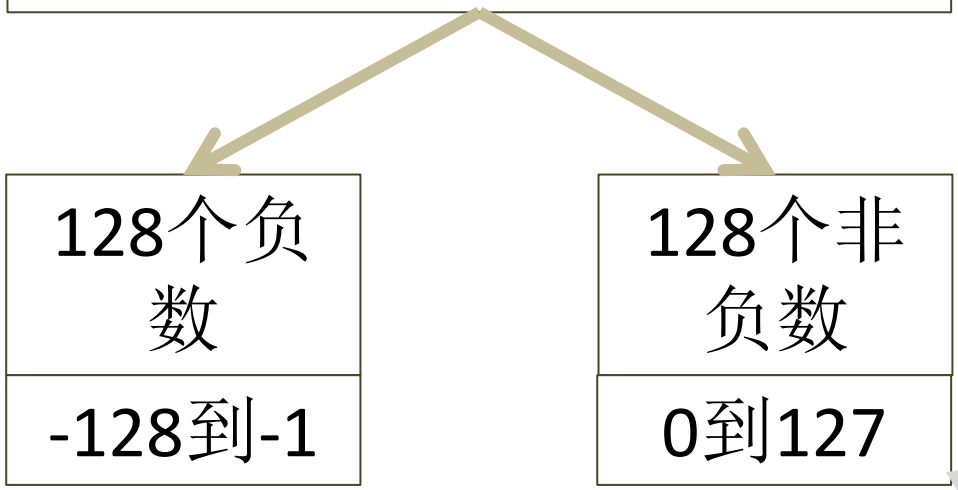
数据结构

补码

为什么用补码?

如果用8位二进制来表示一个整数
可以有2^8=256种编码可能性

正数和负数都需要编码
这256种编码被一分为二



为什么用补码?

256种编码代表
最小数从-128到最大数127
之间的这256个数

这意味着128到255这些数字没有分配到编码。本应该属于128到255的二进制编码被分配给了负数们。具体编码的规律如下：

编码00000000代表的整数是0
编码00000001代表的整数是1
编码00000010代表的整数是2
……
编码01111111代表的整数是127

编码00000000代表的整数是0
编码00000001代表的整数是1
编码00000010代表的整数是2
……
编码01111111代表的整数是127
编码10000000代表的整数是-128(而不是128)
编码10000001代表的整数是-127(而不是129)
……
编码11111111代表的整数是-1(而不是255)

补码计算

使用八位二进制补码来表示 -128到+127 的 共256个整数

用纸和笔计算
10011111代表哪个数?

二进制10011111对应十进制的
128+16+8+4+2+1 = 159
但159太大,超过了127,
所以159要把这个编码让给对应负数
159-256 = -97

补码计算

使用八位二进制补码来表示 -128到+127 的 共256个整数

用纸和笔计算
10111101代表哪个数?

二进制10111001对应十进制的
128+32+16+8+1 = 185
但185太大,超过了127,
所以185要把这个编码让给对应负数
185-256 = -71

补码计算

输入一个8位二进制的编码，若用补码规则，这个编码代表的数字是几？

输入： 01111111
输出： 127

输入： 11111111
输出： -1

算法步骤
二进制转十进制
判断十进制数是否超过127
决定是否是负数

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
int main(){
    string s;
    cin>>s;
    ll ans=0;
    for(ll i=0;i<s.size();i++)
        ans=ans*2+s[i]-'0';
    if(ans>=128)
        ans-=256;
    cout<<ans<<endl;
    return 0;
}
```

程序填空

int补码

1个**int** 占用4个字节
也就是32个二进制位

int类型整数1在计算机中用二进制表示
00000000000000000000000000000001

int类型整数3在计算机中用二进制表示
00000000000000000000000000000011

int类型整数-1在计算机中用二进制表示
是什么样子呢?

11111111111111111111111111111111

二进制 位运算

~

按位 取反

~0是1，~1是0

~00000000000000000000000000000001
就是11111111111111111111111111111110

~11011111101111011111101111101100
就是00100000010000100000010000010011

预测结果

```
int x=1;
int y=2;
cout<<(~x)<<endl;
cout<<(~y)<<endl;
```

```
int a=-1;
int b=-2;
cout<<(~a)<<endl;
cout<<(~b)<<endl;
```

&

按位 与

1&1是1，0&0是0
1&0是0，0&1是0

11011111101111011111101111101100
&
10111100101100111110001000001100
得到
10011100101100011110001000001100

预测结果

```
int x=1;
int y=2;
cout<<(x&y)<<endl;
```

```
int a=7;
int b=5;
cout<<(a&b)<<endl;
```

按位 或

1|1是1, 0|0是0
1|0是1, 0|1是1

11011111101111011111101111101100

|

10111100101100111110001000001100

得到

11111111101111111111101111101100

预测结果

```
int x=1;
int y=2;
cout<<(x|y)<<endl;
```

```
int a=9;
int b=3;
cout<<(a|b)<<endl;
```

^

按位 异或

1^1是0，0^0是0
1^0是1，0^1是1

11011111101111011111101111101100

^

10111100101100111110001000001100

得到

01100011000011100001100111100000

预测结果

```
int x=1;
int y=2;
cout<<(x^y)<<endl;
```

```
int a=7;
int b=3;
cout<<(a^b)<<endl;
```

```
cout<<(a xor b)<<endl;
```

另类理解

异或就是不进位的加法

异或就是不借位的减法

^

按位 异或

1^1是0，0^0是0

1^0是1，0^1是1

11011111101111011111101111101100

^

10111100101100111110001000001100

得到

01100011000011100001100111100000

预测结果

```
int x=1;
int y=2;
cout<<(x^y)<<endl;
```

```
int a=7;
int b=3;
cout<<(a^b)<<endl;
```

```
cout<<(a xor b)<<endl;
```

优先级

位运算的优先级非常低
建议位运算都使用括号

预测结果

```
4    cout<<(1+(1&1))<<endl;
5    cout<<(1+1&1)<<endl;
```

```
6    cout<<(1+(1|1))<<endl;
7    cout<<(1+1|1)<<endl;
```

<<

二进制左移1位
右侧补零

1<<1是2, 2<<1是4
3<<1是6, 5<<4是80

11111110000000000001111111000100
<<4
得到
11100000000000011111110001000000

预测结果

```
cout<<(1<<3)<<endl;
cout<<(1<<4)<<endl;
cout<<(2<<1)<<endl;
cout<<(2<<2)<<endl;
```

```
cout<<(3<<1)<<endl;
cout<<(3<<2)<<endl;
cout<<(4<<1)<<endl;
cout<<(5<<2)<<endl;
```

速度更快

左移k位：乘以2^k

<<

二进制左移1位
右侧补零

1<<1是2, 2<<1是4
3<<1是6, 5<<4是80

11111110000000000011111111000100
<<4
得到
11100000000000111111100010000000

预测结果

```
cout<<(1<<3)<<endl;
cout<<(1<<4)<<endl;
cout<<(2<<1)<<endl;
cout<<(2<<2)<<endl;
```

```
cout<<(3<<1)<<endl;
cout<<(3<<2)<<endl;
cout<<(4<<1)<<endl;
cout<<(5<<2)<<endl;
```

>>

二进制右移1位
左侧补零

2>>1是1, 4>>1是2
7>>1是3, 11>>2是2

11111110000000000001111111000100
>>4
得到
00001111111000000000000111111100

预测结果

```
cout<<(1>>3)<<endl;
cout<<(1>>4)<<endl;
cout<<(2>>1)<<endl;
cout<<(2>>2)<<endl;
```

```
cout<<(3>>1)<<endl;
cout<<(3>>2)<<endl;
cout<<(4>>1)<<endl;
cout<<(5>>2)<<endl;
```

速度更快

右移k位：除以2^k取整

>>

二进制右移1位
左侧补零

2>>1是1, 4>>1是2
7>>1是3, 11>>2是2

11111110000000000011111111000100
>>4
得到
00001111111000000000000111111100

预测结果

```
cout<<(1>>3)<<endl;
cout<<(1>>4)<<endl;
cout<<(2>>1)<<endl;
cout<<(2>>2)<<endl;
```

```
cout<<(3>>1)<<endl;
cout<<(3>>2)<<endl;
cout<<(4>>1)<<endl;
cout<<(5>>2)<<endl;
```

优先级

位运算的优先级非常低
建议位运算都使用括号

预测结果

```
4    cout<<(1+1<<1)<<endl;
5    cout<<(1+(1<<1))<<endl;

6    cout<<(2+4>>1)<<endl;
7    cout<<(2+(4>>1))<<endl;
```

```
int lowbit(int a){
    return a&(-a);
}
```

表示a的二进制表示中最低位的1代表是十进制的几

十进制	二进制
0	00000000
a	10110100
0-a	01001100
a&(-a)	00000100

lowbit(1)是1
lowbit(2)是2
lowbit(4)是4
lowbit(6)是2
lowbit(12)是4
lowbit(40)是8

为了统计一个非负整数的二进制形式中 1 的个数，代码如下：

```
int CountBit(int x) {
    int ret = 0;
    while (x) {
        ret++;
        _____;
    }
    return ret;
}
```

空格内要填入的语句是(　　)

A) x >>= 1

B) x &= x - 1

C) x |= x >> 1

D) x <<= 1

为了统计一个非负整数的二进制形式中 1 的个数，代码如下：

```
int CountBit(int x) {
        int ret = 0;
        while (x) {
            ret++;
                ____________;
        }
        return ret;
}
```

空格内要填入的语句是(　　)

A) x >>= 1
B) x &= x - 1
C) x |= x >> 1
D) x <<= 1

十进制	二进制
0	00000000
x	10110100
x-1	10110011
x&(x-1)	10110000

去掉末尾的1

01个数

输入十进制正整数n，输出n的二进制共有几位？其中有几个1?有几个0?
输入一个正整数n。n<=10^{10}
输出3个整数，由空格隔开。

输入： 2
输出： 2 1 1

输入： 6
输出： 3 2 1

算法步骤
不断循环重复
取出二进制末尾 删除二进制末尾

01个数

```
 5    ll n,c1=0,c0=0;
 6    cin>>n;
 7    while(n){
 8        if(n&1) c1++;
 9        else c0++;
10        n=n>>1;
11    }
```

跟着老师翻译
理解每一行

太戈编程

715.位运算lowbit

719.位运算01计数

拓展题	723.补码转整数
拓展题	722.整数转补码