第二次上机解析

A 字节清零

难度	考点
1	位运算

题目分析

解法1

为了消去输入数据的从右往左数的第二个8位，可以将它和 11111111 11111111 00000000 11111111 求位与。在C语言中，你不必将这个数字算出来，只要在程序里写0xFFFF00FF（这会是一个unsigned int)就可以了。

解法2

我们可以采用左移与按位或来得到0xFFFF00FF。

示例代码

解法1

```
#include <stdio.h>

int main()
{
    int a, b;
    scanf("%d", &a);
    b=a&0xFFFF00FF;
    printf("%d", b);
    return 0;
}
```

解法2代码片段

```
int n, a = ( (1<<8) | (1<<9) | (1<<10) | (1<<11) | (1<<12) | (1<<13) | (1<<14) | (1<<15));
scanf("%d", &n);
printf("%d", n & (~a));
```

```
int n, a = 0, i;
for (i = 8; i <= 15; i++) a |= (1<<i);
scanf("%d", &n);
printf("%d", n & (~a));
```

B 最萌身高差

难度	考点
2	双重循环

题目分析

这是一道简单题。本题只考察双重循环，而几乎没有内置任何其余操作。学生只需通过简单的双重循环遍历数组，找出所有满足条件的数对即可。

示例代码

```
#include<stdio.h>

int main() {
    int N, h[100], i, j;
    scanf("%d", &N);
    for (i = 0; i < N; i++) {
        scanf("%d", &h[i]);
    }
    for (i = 0; i < N - 1; i++) {
        for (j = i + 1; j < N; j++) {
            if ((h[j] - h[i]) % 2 == 0 && (h[j] - h[i] > 10 || h[i] - h[j] > 10)) {
                printf("%d %d\n", i + 1, j + 1);
            }
        }
    }
    return 0;
}
```

C 求阶乘

难度	考点
3	循环、求余

题目分析

主要考察使用循环求阶乘，注意0是一个特殊情况，要单独考虑。示例代码中因为定义了$prod$的初值为1，0的时候不进循环，所以对结果并没有影响。

数据范围会超出int的范围，所以在每一次的计算过程中都要求余，最后输出$prod$即可。

值得注意的是，求模运算和乘法、加法可以任意嵌套，即都乘完或加完再模和边乘边模的结果是一样的。

示例代码

```
#include<stdio.h>
int main()
{
    int n,i;
    int prod=1;
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        prod*=i;
        prod=prod%1000033;
    }
    printf("%d\n",prod);
    return 0;
}
```

D 宋老师的进制转化

难度	考点
3	数学计算

题目解析

将一个数字变成3进制，只需要对它逐步模三取余再除以三直到它变成0，再倒序输出即可。

示例代码

```
#include <stdio.h>

int main(){
    int a;
    int ans[100],num,i;
    while (scanf("%d",&a)>0)
    {
        num=0;
        ans[num]=a%3;
        a/=3;
```

```
            num++;
            while(a>0)
            {
                ans[num]=a%3;
                a/=3;
                num++;
            }
            if(num>=5){
                printf("LONG");
            }
            for(i=num-1;i>=0;--i){
                printf("%d",ans[i]);
            }
            printf("\n");
        }
        return 0;
    }
```

E 统计单词数

题目解析

从编写的角度讲，单词数=空格数+1，所以有

示例代码

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

int main() {
    int n;
    char c;
    while((c=getchar())!= '.')
    {
        if(c==' ') n++;
    }
    printf("%d",++n);
    return 0;
}
```

F 超大统计

难度	考点
1	long long型的运用，不定组数输入

题目分析

这题比较简单，出题的意图在于使同学们了解long long型的用法及数据范围，提升大家对于变量范围的敏感性。

示例代码

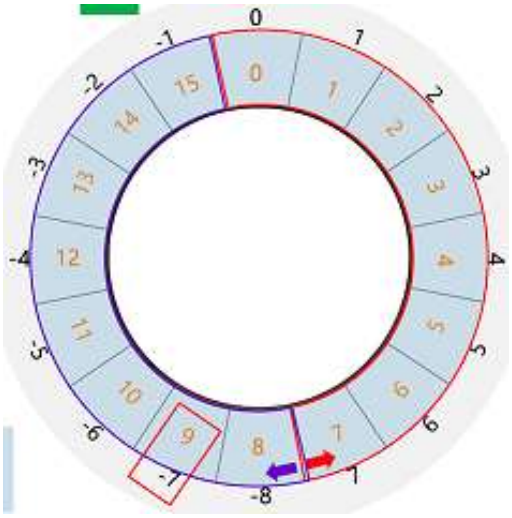
```
#include <stdio.h>
int main()
{
        int i;
        long long num,sum=0;

        while(~scanf("%lld",&num)){
                sum+=num;
        }

        printf("%lld",sum);

        return 0;
}
```

G W位数

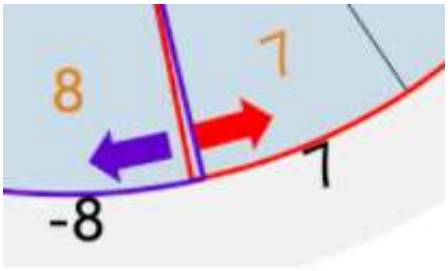
难度	考点
3	数据类型、位运算

题目解析

补码是利用了"补数"的概念将一个w位的二进制编码空间一一对应到了$[-2^{w-1}, 2^{w-1}-1]$的整数空间。以$w=4$为例，补码的映射规则可以形象地使用老师PPT中的这个圈来表示。



但是如果我们把一个过大的数字强行存入这个4位补码数字，会发生什么情况？也就是溢出，



4位补码数字下，-9会溢出为7（上图红色箭头方向），8会溢出为-8（上图紫色箭头方向）。数字溢出了多少，就在这个图中的圈上沿着溢出方向多走几格。由于这是个圈，我们可以采用求余的方式计算这个溢出问题。下文中的 $\%$ 等效于C语言中的 `%` 求余运算符。

我们将一个整数x强行存入一个w位二进制补码数b：

如果整数x在w位二进制补码表示方式下发生了上溢出（太大），则b的实际值为：

$$b = (x - (2^{w-1} - 1) - 1) \% 2^w + (-2^{w-1})$$

如果整数x在w位二进制补码表示方式下发生了下溢出（太小），则b的实际值为：

$$b = (x - (-2^{w-1}) + 1) \% 2^w + (2^{w-1} - 1)$$

- 示例代码1中 `m = 1LL << w` 可能会导致溢出，但是其实并不影响运算结果，可以思考一下原因。

另一种使用位运算的做法是利用了计算机在处理溢出的时候，实际上是将多余的高位二进制数字直接舍去。换言之，只要截取x的二进制表示的后w位即可，做法即示例代码2中的

```
sum = (sum << (64 - w)) >> (64 - w);
```

其中64是 `long long int` 的位数长度。该做法涉及到算术右移（补符号位），下面解释一下什么是逻辑右移。

对于左移，低位补零这一点没有问题，但是对于右移来说，无符号数右移一定是在高位补零；有符号数

会在高位补符号位，也就是非负数补零，负数补一：

```
int main()
{       //char是有符号数
    char a = 100;   //01100100
    char b = -100;  //10011100
    a >>= 3;        //01100100 -> 00001100 =  12 =  100 / 2^3
    b >>= 3;        //10011100 -> 11110011 = -13 = -100 / 2^3 - 1
}
```

对我们做这道题来说，算术右移帮助我们补上符号位，使得答案是负值的情况下我们能够使用 long long 类型来正确输出一个W位数，不然我们就要使用类似这样的手段来处理 sum：

```
long long mask = 0;
mask = ~(~mask << w);
long long ans;
if (( sum & (1LL << (w - 1)) ) == 0)
{
    ans = sum & mask;
}
else
{
    ans = -((~sum + 1) & mask);
}
```

为的是回避使用掩码截取了后w位后，W位数的“首位”符号位是1，但是在它更前面的高位全是0， long long 类型的数据不会认为它是个负值的问题，我们就需要采取一点手段来得到的正确结果。但如果使用了算术右移，我们便可以一步到位。

- 看到有很多同学采取这样的写法：`( ( (a+b)%(1LL<<w) )<<(64-w) )>>(64-w)`，这里的 `%(1LL<<w)` 是完全多余的。

示例代码1

```
#include <stdio.h>
int main()
{
    long long int a, b, sum = 0, l, r, m, w;
    //%hdd现在表示w位的二进制补码数字
    while (~scanf("%lld %lld %lld", &a, &b, &w))
    {
        sum = a + b;
        l = -(1LL << (w - 1)), r = 1LL << (w - 1), m = 1LL << w;
        if (sum < l)  //下溢出
        {
            sum = (sum - l + 1) % m + r - 1;
        }
        else if (sum >= r)  //上溢出
        {
            sum = (sum - r) % m + l;
```

```
        }
        printf("%lld + %lld = %lld\n", a, b, sum);
    }

    return 0;
}
```

示例代码2

```
#include <stdio.h>
int main()
{
    long long a, b, w, sum;
    while (~scanf("%lld %lld %lld", &a, &b, &w))
    {
        sum = a + b;
        sum = (sum << (64 - w)) >> (64 - w); //一行
        printf("%lld + %lld = %lld\n", a, b, sum);
    }

    return 0;
}
```

H 弹反

难度	考点
2	数学计算、模拟

题目解析

将一个10位以内的整数n翻转，要求正负性，不含前导0。

本题有数学运算和字符数组两种写法，前者考虑利用取模运算由低到高提取n的每一位并输出，后者直接做简单的判断和倒序输出即可。

示例代码1

```
#include <stdio.h>
int main()
{
    long long n, ans = 0;
    scanf("%lld", &n);
    while (n) // n为0时跳出循环
    {
        ans = ans * 10 + n % 10; // n % 10 提取当前n的末位，若n < 0，则n % 10也为负数
        n /= 10; // 舍弃n的最低位
```

```
    }
    printf("%lld", ans);
    return 0;
}
```

示例代码2

```
#include <stdio.h>
#include <string.h>
char str[15];
int main()
{
    int i, len = 0, flag1 = 0, flag2 = 0;
    scanf("%s", str);
    len = strlen(str); //获取字符串长度
    if (len == 1) //若为0，直接输出并结束程序
    {
        printf("%c",str[0]);
        return 0;
    }
    if (str[0] == '-') //若为负数，直接输出符号，且倒序输出时不需遍历到第一位
    {
        printf("%c",str[0]);
        flag1 = 1;
    }
    for (i = len - 1; i >= flag1; i--)
    {
        if (flag2 == 0 && str[i] == '0') //若此前从未输出过0，则置之不理
            ;
        else
        {
            flag2 = 1; //代表以后可以输出0了
            printf("%c",str[i]);
        }
    }
    return 0;
}
```

I A op B Problem

难度	考点
4	位运算

题目分析

本题的难点在于将二进制中某一位提取出来，并将答案赋值为某个特定值。根据题目和课件中的提示，不难想到下述方法：

`x & (1 << (i - 1))`：从低位到高位，提取二进制数x的第i位（下标从1开始）。若该位为1，则表达式的结果为2^{i-1}，否则为0。
`x = x | (w << (i - 1))`：从低位到高位，将二进制数x的第i位（下标从1开始）设为w，其中$w \in \{0, 1\}$。

同时注意到数据范围在$[0, 2^{32})$，因此需要采用 `unsigned int` 或是 `long long` 完成相关任务。

示例代码1

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

// 合理使用宏定义能够方便代码编写
#define ui unsigned int

int main(){
    int q;
    ui a, b, a0, b0;
    ui w0, w1, w2, w3;
    ui ans;
    int i;

    scanf("%d", &q);
    while(q--){
        scanf("%u%u", &a, &b);  // unsigned int 使用%u输入输出
        scanf("%u%u%u%u", &w0, &w1, &w2, &w3);

        // 每次都要重置 ans
        ans = 0;
        for(i = 0; i < 32; i++){
            a0 = a & (1 << i);      // 提取 a 的第 i 位
            b0 = b & (1 << i);      // 提取 b 的第 i 位
            if(a0 == 0 && b0 == 0) ans |= w0 << i;
            else if(a0 == 0 && b0 > 0) ans |= w1 << i;
            else if(a0 > 0 && b0 == 0) ans |= w2 << i;
            else ans |= w3 << i;
        }

        printf("%u\n", ans);
    }

    return 0;
}
```

示例代码2

利用二进制数的性质，我们可以将a_0, b_0的值看成一个二位二进制数上的两位，每次只需要根据a_0, b_0的值，选择对应的w_i作为结果。因此可以有如下代码：

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

```c
#define ui unsigned int

int main(){
    int q;
    ui a, b, a0, b0;
    ui ans, x;
    ui w[4];
    int i;

    scanf("%d", &q);
    while(q--){
        scanf("%u%u", &a, &b);
        for(i = 0; i < 4; i++)
            scanf("%d", &w[i]);

        ans = 0;
        for(i = 0; i < 32; i++){
            a0 = a & (1 << i);
            b0 = b & (1 << i);
            // 将 a0 与 b0 合成一个二进制数 x
            x = (a0 != 0) * 2 + (b0 != 0);
            ans |= w[x] << i;
        }

        printf("%u\n", ans);
    }

    return 0;
}
```

示例代码3

下面的解法是由翁韬涵（19374340）同学提供的一种同样很巧妙的做法。这种做法利用了位运算的性质，减少了每次询问枚举每一位的循环时间，时间复杂度上优于上述两种做法。

该做法的正确性不在此详述，学有余力的同学可以思考为什么这样做可以得到答案。

```c
// 代码经过一定规范性调整，并非真实原代码
#include <stdio.h>
#include <stdlib.h>

int main(){
    int q;
    int w[4], i;
    unsigned int a, b, sum;
    unsigned int m, m1, m2, m3;

    scanf("%d", &q);
    for(i = 0; i < q; i++){
        sum = 0;                // 每次循环重置 sum
        scanf("%u%u", &a, &b);
        scanf("%d%d%d%d", &w[0], &w[1], &w[2], &w[3]);
        m = a ^ b;
```

```
            m1 = m & b;
            m2 = m & a;
            m3 = a & b;
            sum = m1 * w[1] + m2 * w[2] + m3 * w[3];
            printf("%u\n", sum);
        }

        return 0;
    }
```

J 求相反数

难度	考点
3	位运算

题目解析

解法1

求一个正数的相反数的补码的步骤如下： 1. 将首位从0边成1（得到相反数的原码） 2. 将后面几位取反（得到相反数的反码） 3. 给数字+1（得到相反数的补码）

容易发现，前两步可以变为一步，即对整个数字取反后+1即得答案。 要从负数得到相反数的补码只需将上面两步反过来。即： 1. 数字-1 2. 取反

注意在这个过程中会出现数字溢出（补码的负数能比正数多表示一个），因此我们还需要判断是否溢出。最大负整数的补码都形如 `10000` ,在减1后变成 `01111` ，即首位变成了0，所以可以利用这个特点把溢出判定放在1、2两步之间。

解法2

我们以四位数为例， `x=1010` 是一个补码，则 `~x=0101` ， 易知 `x+~x=-1` ，这个对任何x都是成立的。所以有 `-x=~x+1` 对任何补码均成立。

示例代码

解法1的代码

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

char s[100005] = {0};  // 大数组要定义为全局

int main()
{
    int len, i, j, flag;
```

```
while (scanf("%s", s) > 0)
{
    len = strlen(s);
    if (s[0] == '0')
    { // 正数
        // 取反
        for (i = 0; i < len; ++i)
        {
            s[i] = s[i] == '0' ? '1' : '0';
        }
        // 加一
        flag = 1;     //用于判断是否需要进位
        for (i = len - 1; i >= 0; --i)
        {
            if(flag)
            {
              if(s[i] == '1')
                {
                 s[i] = '0';
                 flag = 1;
                }
              else
                {
                 s[i] = '1';
                 flag = 0;
                }
            }
        }
        printf("%s\n",s);
    }
    else // 负数
    {
        // 减1
        flag = 1;   //用于判断是否需要借位
        for (i = len - 1; i >= 0; --i)
        {
            if(flag)
            {
             if(s[i] == '1')
              {
               s[i] = '0';
               flag = 0;
              }
             else
               {
               s[i] = '1';
               flag = 1;
               }
             }
        }

        // 判断溢出
        if (s[0] == '0')
        {
            printf("0verf1ow! \n");
        }
```

```
            else
            {
                //取反
                for (i = 0; i < len; ++i)
                {
                    s[i] = s[i] == '0' ? '1' : '0';
                }
                printf("%s\n",s);
            }
        }
    }
    return 0;
}
```

解法2的代码

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

char bin[100007];

int main()
{
    int i, len, flag, count;
    while (~scanf("%s", bin))
    {
        len = strlen(bin);
        for (i = 0; i < len; i++)
        {
            bin[i] = bin[i] == '0' ? '1' : '0';
        }
        flag = 1;     //用于判断是否需要进位
        count = 0;    //用于记录进位次数
        for (i = len - 1; i >= 0; i--)
        {
          if(flag)
          {
             if(bin[i] == '1')
              {
                bin[i] = '0';
                flag = 1;
                count++;
              }
             else
              {
                bin[i] = '1';
                flag = 0;
              }
          }
        }
        if (count == len-1)
        {
            printf("0verf1ow!\n");
        }
        else
```

```
        {
            printf("%s\n", bin);
        }
    }

    return 0;
}
```

K 我家大小姐不想让我告白

难度	考点
2	循环移位

题目分析

这道题对于学习过循环移位的同学来说不难，没有学习过循环移位的同学可以参考HINT针对每个二进制位进行操作，也可以做。

示例代码

```
#include<stdio.h>
int main()
{
    int i,N,n,m;
    unsigned int a;
    scanf("%d%d",&m,&N);
    for(i=0;i<m;++i)
    {
        scanf("%d%d",&n,&a);
        printf("%d\n",((a>>(N-n))|(a<<n))%(1<<N));
    }
    return 0;
}
```

L TaoFu算进制转换

难度	考点
5	进制转换

题目分析

辗转相除法，没什么好说的。注意大于9的数字转换为字母即可。

下面有用递归的做法，核心思想是一样的。

示例代码1

```
#include <stdio.h>
int main()
{
    int a,k;
    char num[33]={0};
    scanf("%d %d",&a,&k);
    int q,i=-1;
    q=a/k;                          //求商
    num[++i]='0'+a%k;               //求余数
    a=q;              //商作为下一次运算的被除数
    while(a!=0)
    {
        q=a/k;                          //求商
        num[++i]='0'+a%k;               //求余数
        a=q;                            //商作为下一次运算的被除数
    }
    for(;i>=0;i--)
    {
        if(num[i]>'9') num[i]+=39;
        printf("%c",num[i]);
    }
    return 0;
}
```

示例代码2

```
#include <stdio.h>

int main() {
    int a,k,i,t[33];
    char tab[]={"0123456789abcdef"};//事先将可能的输出按照顺序列表
    scanf("%d%d",&a,&k);
        if(a==0)
           printf("0");
        else
        {
           for(i=1;a!=0;i++)
            {
            t[i]=a%k;
            a=a/k;
            }
        while(--i)
           printf("%c",tab[t[i]]);//将余数当作索引输出列表tab中的元素
        }
    return 0;
}
```