

目 录

1 求一个集合中的连续串，使得这个连续串中各个数相加的和最大	1
2 求一个集合中的连续串，使得这个连续串中各个数相加的和最小	2
3 动态规划求组合	3
4 寻找发贴“水王”	4
5 求一个字符串中最长的重复子串，'0000 '不算在内	6
6 求两个字符串的最长公共子串	8
7 读一组整数到 VECTOR 对象，计算并输出每对相邻元素的和。如果读入元素个数为奇数， 则提示用户最后一个元素没有求和，并输出其值。然后修改程序：头尾元素两两配对（第一个 和最后一个，第二个和倒数第二个，以此类推），计算每对元素的和，并输出。	9
8 整数转化为字符串	12
9 字符串转化为整数	13
10 转换字符串格式为：原来字符串里的字符 + 该字条连续出现的个数，例如字符串： 1233422222 转化为 1121324125 （1 出现 1 次，2 出现 1 次，3 出现 2 次.....） ...	14
11 将一句话里的单词进行倒置，标点符号不倒换。比如 “ I COME FROM BEIJING. ” “ BEIJING. FROM COME I ”	15
12 二叉树根结点为 ROOT，用递归法把二叉树的叶子结点按从左到右的顺序连成一个单链表	17
13 连续正整数之和	17
14 文件中有一组整数，要求排序后输出到另一个文件中	19
15 小猪吃米	21
16 在一个数组中存在着新数组，求出新数组的长度。	23
17 写函数找出一个字符串中出现频率最高的字符（如果最高频率相同的有多个字符，取最先 遇见的那个字符）	24
18 十三个人围成一个圈，从第一个人开始顺序报号 1、2、3。凡是报到“3”者退出圈子，请 找出最后留在圈子中的人原来的序号	26
19 已知 N 个人（以编号 1，2，3...N 分别表示）围坐在一张圆桌周围。从编号为 K 的人开 始报数，数到 M 的那个人出列；他的下一个人又从 1 开始报数，数到 M 的那个人又出列；依 此规律重复下去，直到圆桌周围的人全部出列	28
20 十进制正数或负数转化为二进制	31
21 将阿拉伯数字转化为中文数字，如 12 “一十二”	32
22 大数存储，求 100 的阶乘	35
23 在一个字符串中找到第一个只出现一次的字符。如 “ ABACCDEFF ”，输出 B。	36

1 求一个集合中的连续串，使得这个连续串中各个数相加的和最大

```
#include<stdio.h>

int getmax(int a[], int n, int *begin, int *end);

into main(void)
{
    int a[] = {-1,-2,-3,100,-4,-5,6,-7,9,200};
    int begin;
    int end;
    int sum;
    sum = getmax(a,10,&begin,&end);
    printf("The maximal sum is %d\n",sum);
    printf("The begin index is %d, the end index is %d\n",begin,end);
    return 0;
}
```

/*++

算法：

从第一个数出发，向右叠加，将他们的和累加于 sum，只要和大于零，就继续。

期间，保存这些和值中的最大值为 max。如果 sum 小于零，则从 sum 小于零的后一个重新计算。

--*/

```
int getmax(int *a, int n, int *begin, int *end)
{
    int max = a[0];
    int sum = a[0];
    int tempbegin = 0;
    *begin = 0;
    *end = 0;
    for (int i = 1; i < n; i++)
    {
        if(sum > 0)
        {
            sum += a[i];
        }
        else
        {
            tempbegin = i;
            sum = a[i];
        }
    }
}
```

```

        if (max <= sum)
        {
            max = sum;
            *begin = tempbegin;
            *end = i;
        }
    }
    return max;
}

```

```

The max sum is 299
The begin index is 3, the end index is 9

```

2 求一个集合中的连续串，使得这个连续串中各个数相加的和最小

```

#include <stdio.h>
int getmin(int a[], int n, int *begin, int *end);

int main(void)
{
    // 测试数组全部通过测试
    // int a[] = {8, 9, -3, -10, 7, 0, 8, -12, 9, 8, -1, -2, 9};
    // int a[] = {1, 2, 3, 4, 5};
    // int a[] = {-1, -2, -3, -5, -4};
    // int a[] = {-1, 100, -1000, 100, -1};
    // int a[] = {1, 1, 1, 1, 1};
    // int a[] = {-1, -1, -1, -1, -1};
    // int a[] = {1, -1, 1, -1};
    // int a[] = {8, 9, -3, -10, 7, 0, 8, -12, 9, 8, -1, -2, 9};
    int a[] = {8, 9, -3, -10, 7, -5, 2, -12, 9, 8, -1, -2, 9};
    int begin;
    int end;
    int sum;
    int k = sizeof(a)/sizeof(int);
    sum = getmin(a,k,&begin,&end);
    printf("The minimum sum is %d\n",sum);
    printf("The begin index is %d, the end index is %d\n",begin,end);
    return 0;
}

int getmin(int *a, int n, int *begin, int *end)
{
    int min = a[0];

```

```

int sum = a[0];
int tempbegin = 0;
*begin = 0;
*end = 0;
for (int i = 1; i < n; i++)
{
    if(sum < 0)
    {
        sum = sum + a[i];
    }
    else
    {
        tempbegin = i;
        sum = a[i];
    }

    if (sum <= min)
    {
        min = sum;
        *begin = tempbegin;
        *end = i;
    }
}
return min;
}

```

```

The minimum sum is -21
The begin index is 2, the end index is 7

```

3 动态规划求组合

```
/*+++++++
```

思想： $C(n,k)=C(n-1,k-1)+C(n-1,k)$

利用动态规划法，用一个二维数组把前面算出的组合数

保存起来，这样就不用重复对一个小的组合数算很多次

```
++++++*/
```

```
#include <iostream>
```

```
using namespace std;
```

```
const int N=6;
```

```
const int K=3;
```

```
int a[N+1][K+1]; //N+1 行，K+1 列的存储
```

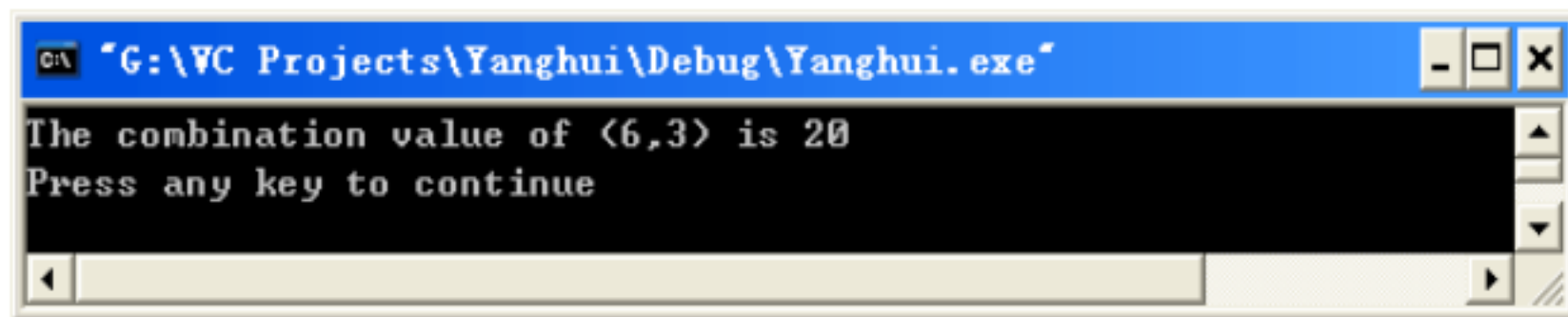
```

int min(int x, int y)
{
    return    x<y?x:y;
}

int Com(int n, int k)
{
    for(int i=0;i<=n;++i)
    {
        for(int j=0;j<=min(i,k;++j)
        {
            if (j==0 || j==i)//        每行第 0 列 (最左边 , C(n,0)=1    , 主对角线元素也为    1)
            {
                a[i][j]=1;
            }
            else
            {
                a[i][j]=a[i-1][j-1]+a[i-1][j];//        根据递推式
            }
        }
    }
    return    a[n][k];
}

void main()
{
    cout<<"The combination value of (6,3) is "<<Com(N,K)<<endl;
}

```



4 寻找发帖 “水王”

1) 题目

一个论坛中有一大 “水王”，他不但喜欢发帖，还会回复其他 ID 发的每个帖子。该 “水王”发帖数目超过了总数的一半。如果你有一个当前论坛所有帖子（包括回帖）的列表，其中帖子作者的 ID 也在表中，你能快速找出这个传说中的 “水王”吗？

2) 分析

如果每次删除两个不同的 ID（不管是否包含“水王”的 ID），那么，在剩下的 ID 列表中，“水王”ID 出现的次数仍然超过总数的一半。看到这一点之后，就可以通过不断重复这个过程，把 ID 列表中的 ID 总数降低（转化为更小的问题），从而得到问题的答案。新的思路，总的时间复杂度只有 $O(N)$ ，且只需要常数的额外内存。

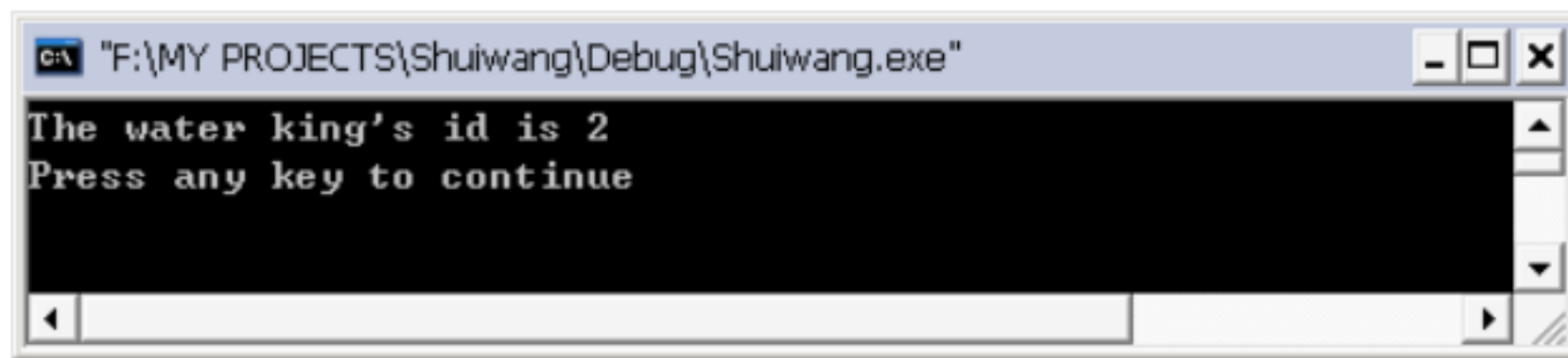
3) 代码

```
/******  
* 功能：寻找发帖“水王”  
*  
* @Author: H          Date: 2009-12-31  
*****/
```

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

```
int Find(int* ID, int N)  
{  
    int candidate;  
    int nTimes=0;  
    int i;  
    for(i = 0; i < N; i++)  
    {  
        if(nTimes == 0)  
        {  
            candidate = ID[i];  
            nTimes = 1;  
        }  
        else  
        {  
            if(ID[i] == candidate)  
                nTimes++;  
            else  
                nTimes--;  
        }  
    }  
    return candidate;  
}
```

```
int main(void)  
{  
    int id[] = {1,2,2,4,2,4,2,2};  
    int iD = Find(id, 8);  
    cout<<"The water king's id is "<<iD<<endl;  
    return 0;  
}
```



4) 结论

在这个题目中，有一个计算机科学中很普遍的思想，就是如何把一个问题转化为规模较小的若干个问题。分治、递推和贪心等都是基于这样的思路。在转化过程中，小的问题跟原问题本质上一致。这样，我们可以通过同样的方式将小问题转化为更小的问题。因此，转化过程是很重要的。像上面这个题目，我们保证了问题的解在小问题中仍然具有与原问题相同的性质：水王的ID 在 ID 列表中的次数超过一半。转化本身计算的效率越高，转化之后问题规模缩小得越快，则整体算法的时间复杂度越低。

5 求一个字符串中最长的重复子串，'0000 ' 不算在内

```

/*****
* 功能：求一个字符串中最长的重复子串，'0000 .....不算在内
*      比如："12334445000006" 的最长子串是 444"
* 日期：2010-1-19
*****/

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

char* GetSubstring(char* strSource)
{
    char* strSubstring; // 用于保存得到的子串，大小在找到最大子串后再确定，作为返回值
    int nLen;           // 源字符串长度
    int nCurPos;        // 当前统计字符串的头指针位置 (相对于原字符串第一个字符的位置)
    int nCurCount;      // 当前统计字符串的长度 (有相同字符组成的子字符串)
    int nPos;           // 当前最长的子串的头指针位置
    int nCount;          // 当前最长的子串的长度

    nLen = strlen(strSource);

    // 初始化变量
    nCount = 1;

```

```

nPos = 0;
nCurCount = 1;
nCurPos = 0;

// 遍历整个字符串
for(int i = 1; i < nLen; i++)
{
    if(strSource[i] == '0')
        continue;
    if(strSource[i] == strSource[nCurPos])//          如果当前字符与子串的字符相同，子串
长度增 1
        nCurCount++;
    else    // 如果不相同，开始统计新的子串
    {
        if(nCurCount > nCount)//    如果当前子串比原先最长的子串长，把当前子串信
息( 起始位置 + 长度 )保留下来
        {
            nCount = nCurCount;
            nPos = nCurPos;
        }
        // 长度复值为 1，新串的开始位置为 i
        nCurCount = 1;
        nCurPos = i;
    }
}

// 为返回的子串分配空间 ( 长度为 nCount, 由于要包括字符串结束符 \0, 故大小要加 1)
strSubstring = (char*)malloc(nCount + 1);

// 复制子串 (用其他函数也可以 )
for(i = 0; i <= nCount; i++)
    strSubstring[i] = strSource[nPos + i];
strSubstring[nCount] = '\0';

return strSubstring;
}

void main()
{
    // 输入一个字符串 strSource
    char *strSource = "123344450000006";
    char* strSubstring = GetSubstring(strSource);

    printf(" 最长子串为 :%s\n  长度为： %d",strSubstring,strlen(strSubstring));
}

```



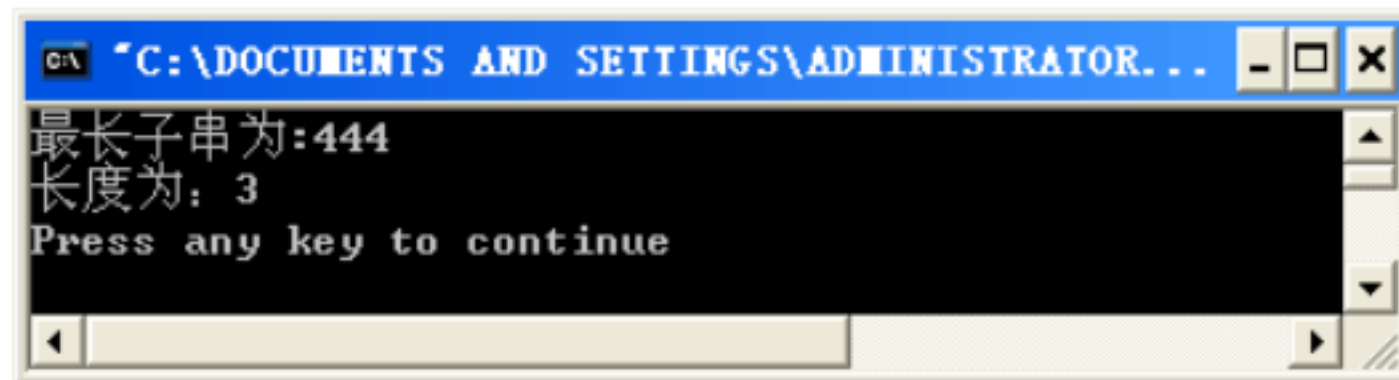
```

printf("\n");

// 释放 strSubstring
free(strSubstring);
}

```

运行结果：



6 求两个字符串的最长公共子串

```

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

```

```

void calculate(char*, char*);

```

```

void main()
{
    char str1[] = "fine,";
    char str2[] = "I am fine.";
    printf(" 两个字符串公共的最长英文单词是      :");
    calculate(str1, str2);
}

```

```

void calculate(char *pp1, char *pp2)
{
    int maxpos = 0, maxlen = 0;           // 记录最长公共子串的位置 ,长度,便于输出
    int len1 = strlen(pp1);
    int len2 = strlen(pp2);
    char *p1 = pp1, *p2 = pp2;

    // 效率上,应该让短串去匹配长串 ,这里让 p1 为短,p2 为长
    if(len1 > len2)
    {
        p1 = pp2;
        p2 = pp1;
        len1 = strlen(pp2);
        len2 = strlen(pp1);
    }
}

```

```

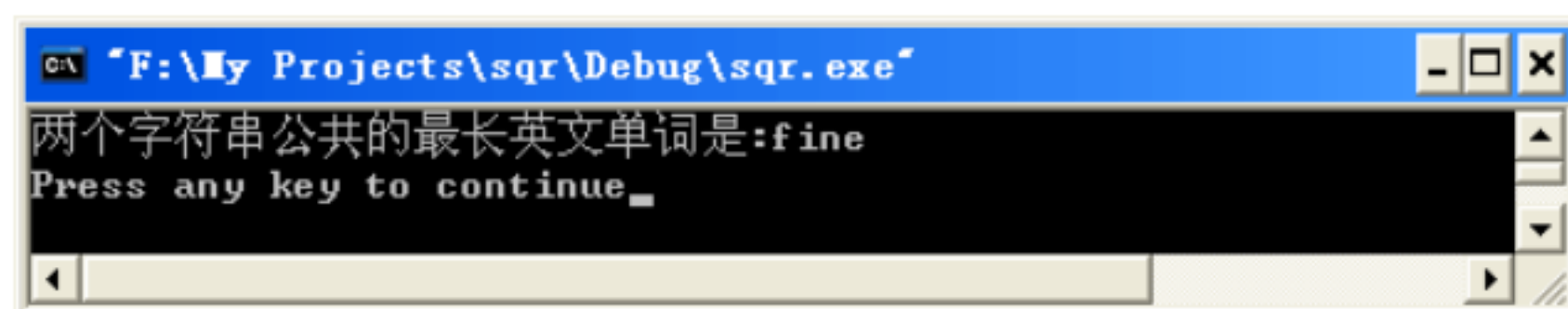
    }

    int i, j, k;
    for(i=0; i<len1; ++i)          // i: 子串头指针在 1 的位置
    {
        for(j=1; j<=len1-i; ++j)    // j: 子串长度
        {
            for(k=0; k<=len2-j; ++k)    // k: 串 2 中待匹配的子串位置
            {
                // 匹配串 1 中 i 开始长度 j 的子串与串 2 中 k 开始长度 j 的子串, 0 表示相等
                if(strncmp(p1+i, p2+k, j) == 0 && j>maxlen)
                {
                    maxpos = i;
                    maxlen = j;          // 记录新的最长子串长度
                }
            }
        }
    }

    // 输出最长公共子串
    for(i=0; i<maxlen; ++i)
    {
        printf("%c", *(p1+maxpos+i));
    }
    printf("\n");
}

```

运行结果：



7 读一组整数到 `vector` 对象，计算并输出每对相邻元素的和。如果读入元素个数为奇数，则

提示用户最后一个元素没有求和，并输出其值。然后修改程序：头尾元素两两配对（第一个和

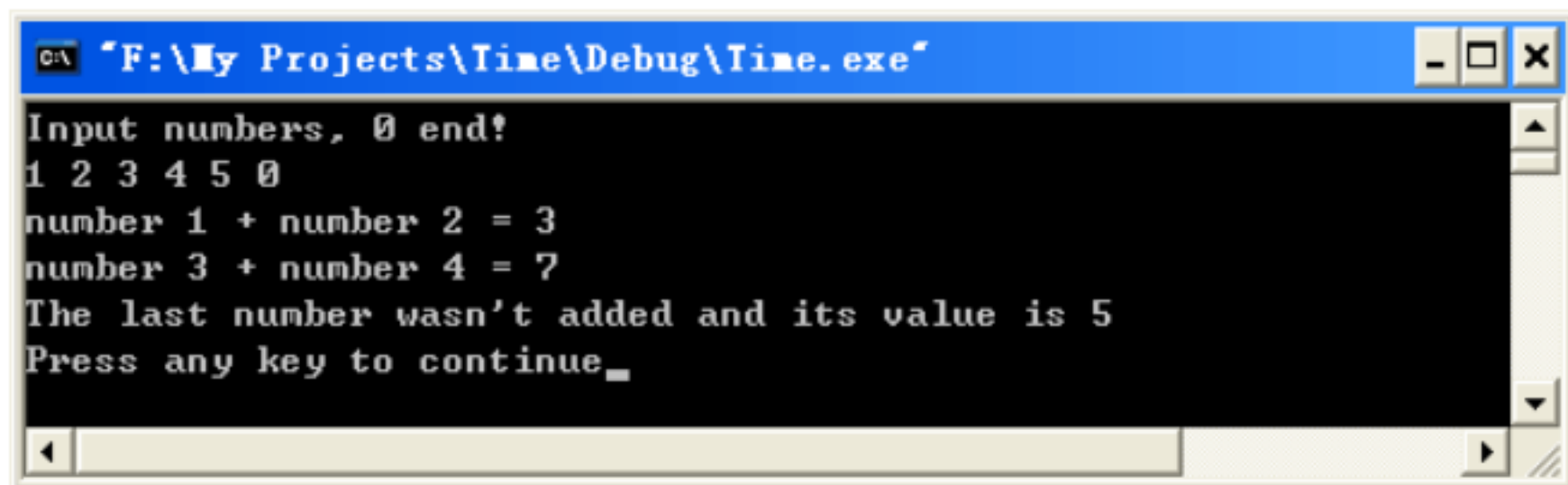
最后一个，第二个和倒数第二个，以此类推），计算每对元素的和，并输出。

解：第一问

```

#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> v;
    int temp, count = 0;
    cout<<"Input numbers, 0 end!"<<endl;
    cin>>temp;
    while(temp)
    {
        v.push_back(temp);
        ++count;
        cin>>temp;
    }
    bool isOdd;
    if(count % 2 == 0)
        isOdd = false;
    else
        isOdd = true;
    if(isOdd)
        --count;
    for(int i = 0; i != count; i += 2)
        cout << "number " << i + 1 << " + number " << i + 2 << " = " << v[i] +
v[i+1] << endl;
    if(isOdd)
    {
        cout << "The last number wasn't added and its value is " << v[v.size() - 1]
<< endl;
    }
    return 0;
}

```



```

C:\ "F:\My Projects\Time\Debug\Time.exe"
Input numbers, 0 end!
1 2 3 4 5 0
number 1 + number 2 = 3
number 3 + number 4 = 7
The last number wasn't added and its value is 5
Press any key to continue_

```

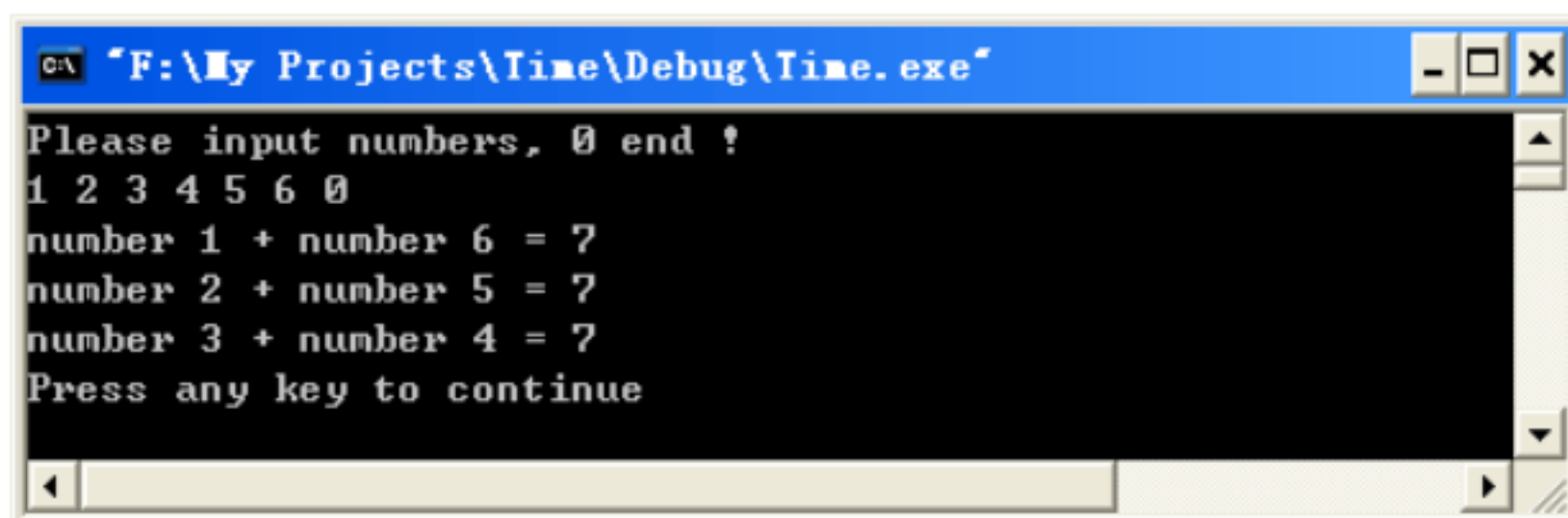
第二问：

```
#include <iostream>
```

```

#include <vector>
using namespace std;
int main()
{
    vector<int> v;
    int temp, count = 0;
    cout<<"Please input numbers, 0 end !"<<endl;
    cin >> temp;
    while(temp)
    {
        v.push_back(temp);
        ++count;
        cin >> temp;
    }
    bool isOdd;
    if(count % 2 == 0)
        isOdd = false;
    else
        isOdd = true;
    for(int i = 0; i != count/2; ++i)
    {
        cout << "number " << i + 1 << " + number " << v.size()-i << " = " << v[i]
+ v[v.size()-1-i] << endl;
    }
    if(isOdd)
    {
        cout << "The middle item wasn't added and its value is: " << v[v.size()/2]
<< endl;
    }
    return 0;
}

```



```

F:\My Projects\Time\Debug\Time.exe
Please input numbers, 0 end !
1 2 3 4 5 6 0
number 1 + number 6 = 7
number 2 + number 5 = 7
number 3 + number 4 = 7
Press any key to continue

```

8 整数转化为字符串

解法一：不用 itoa 函数

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

```
int main(void)
```

```
{
```

```
    int num = 12345;
```

```
    int j = 0;
```

```
    int i = 0;
```

```
    char temp[7], str[7];
```

```
    while(num)
```

```
    {
```

```
        temp[i] = num%10 + '0';
```

```
        i++;
```

```
        num = num/10;
```

```
    }
```

```
    temp[i] = '\0'; //      字符串结束符
```

```
    printf("temp = %s\n", temp);
```

```
    i--;
```

```
    // 上面得到的字符串是逆序的，要反转过来
```

```
    while(i >= 0)
```

```
    {
```

```
        str[j] = temp[i];
```

```
        j++;
```

```
        i--;
```

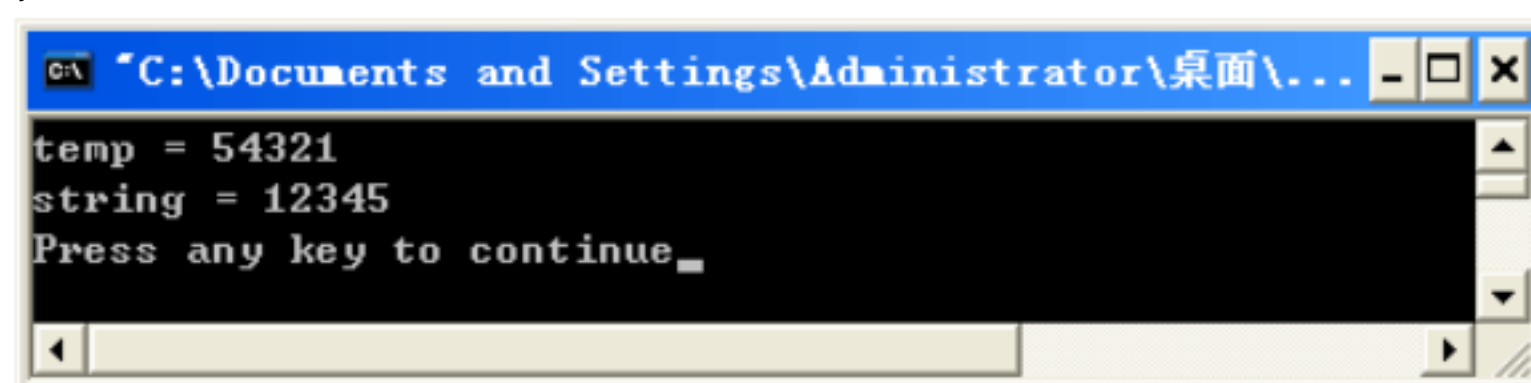
```
    }
```

```
    str[j] = 0; //      字符串结束符
```

```
    printf("string = %s\n", str);
```

```
    return 0;
```

```
}
```



解法二：使用 itoa 函数

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

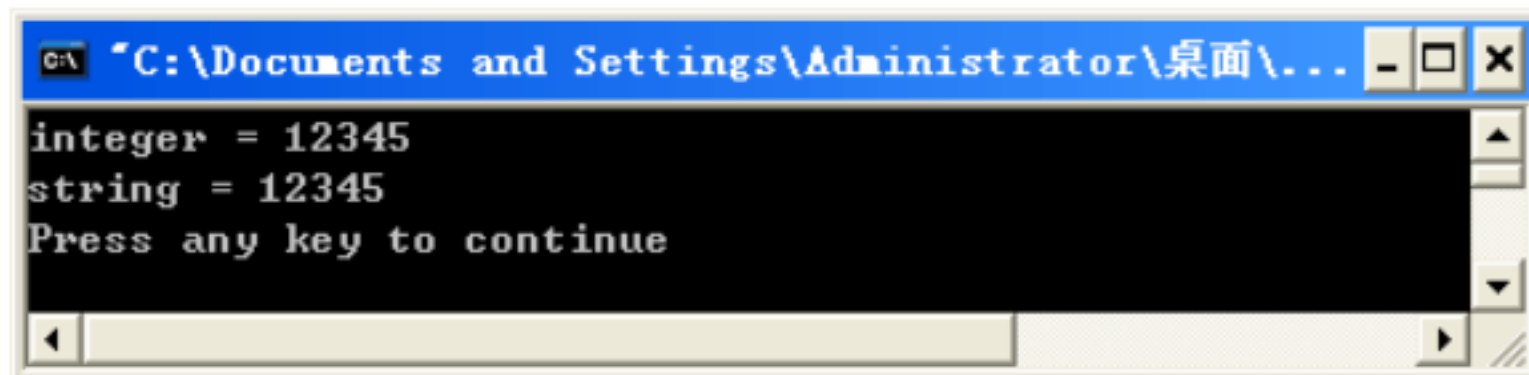
```
#include <iostream>
```

```
int main(void)
```

```

{
    int num = 12345;
    char str[7];
    itoa(num, str, 10);
    printf("integer = %d\nstring = %s\n", num, str);
    return 0;
}

```



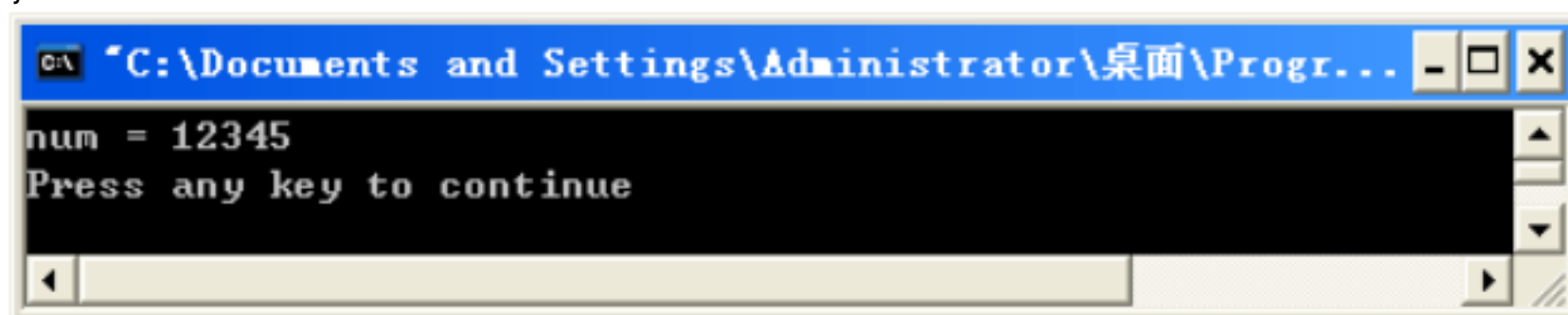
9 字符串转化为整数

```

#include <stdio.h>
#include <iostream>

int main(void)
{
    char *str = "12345";
    int i = 0;
    int num = 0;
    int count = strlen(str);
    while(i < count)
    {
        num = num * 10 + (str[i] - '0');
        i++;
    }
    printf("num = %d\n", num);
    return 0;
}

```



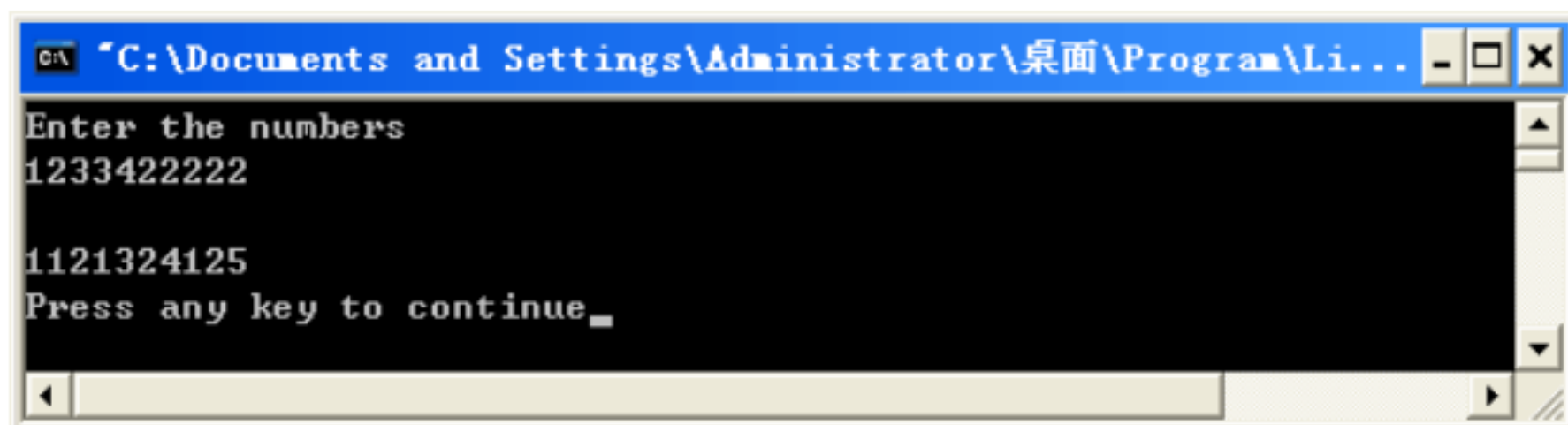
10 转换字符串格式为：原来字符串里的字符

+ 该字条连续出现的个数，例如字符串：

1233422222 转化为 1121324125 (1 出现 1 次，2 出现 1 次，3 出现 2 次.....)

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

int main()
{
    cout<<"Enter the numbers "<<endl;
    string str;
    char retchar[50];
    retchar[0] = '\0';          // 先赋结束符以防止输出乱码
    getline(cin, str);
    int len = str.length();
    int count = 1;
    int k;
    for(k=0; k<=len-1; k++)
    {
        if(str[k+1] == str[k])
        {
            count++;
        }
        else
        {
            sprintf(retchar+strlen(retchar), "%c%d", str[k], count);
            count = 1;
        }
    }
    cout<<retchar<<endl;
    return 0;
}
```



11 将一句话里的单词进行倒置，标点符号不倒换。比如

“ I come from Beijing. ”

“ Beijing. From come I ”

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

int main()
{
    int j = 0;
    int i = 0;
    int begin, end;
    char str[] = "I come from Beijing.";
    j = strlen(str) - 1;
    printf("Origin string = %s\n", str);

    // 第一步进行全盘翻转，将字符串变成 ".gnijieB morf emoc I"
    while(j>i)
    {
        str[i] = str[i] ^ str[j];
        str[j] = str[i] ^ str[j];
        str[i] = str[i] ^ str[j];
        j--;
        i++;
    }
    printf("Temporary string = %s\n", str);

    i = 0;
    // 第二步对每个单词进行转换，如果不是空格，则开始翻转
    while(str[i])
    {
        if(str[i] != ' ')
        {
            begin = i;
            while(str[i] && str[i]!=' ')
            {
                i++;
            }
            i--;
        }
    }
}
```



```

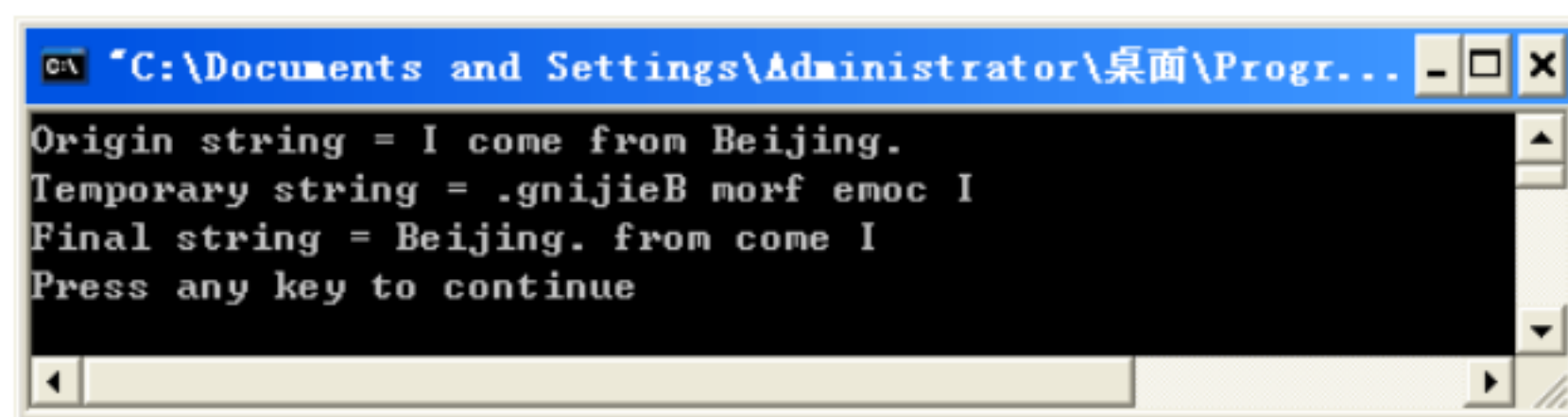
        end = i;
    }

    while(end > begin)
    {
        str[begin] = str[begin] ^ str[end];
        str[end] = str[begin] ^ str[end];
        str[begin] = str[begin] ^ str[end];
        end--;
        begin++;
    }

    i++;
}

printf("Final string = %s\n", str);
return 0;
}

```



The screenshot shows a Windows command prompt window with the title bar "C:\Documents and Settings\Administrator\桌面\Progr...". The window contains the following text:

```

Origin string = I come from Beijing.
Temporary string = .gnijieB morf emoc I
Final string = Beijing. from come I
Press any key to continue

```

The text is displayed in a monospaced font on a black background. The window has standard Windows XP-style window controls (minimize, maximize, close) in the top right corner.

12 二叉树根结点为 root, 用递归法把二叉树的叶子结点按从左到右的顺序连成一个单链表

解：

```
typedef struct node
{
    char c;
    struct node *left,*right;
}btreeNode;
btreeNode *firstLeaf,*pcur; //          分别记录叶子链表的第一个叶子结点及当前结点的前驱

void leafLink(btreeNode *root)
{
    if(!root)
        return;
    if(root->left==NULL && root->right==NULL)
    {
        if(firstLeaf==NULL)
        {
            firstLeaf=root; //          保存找到的第一个叶子结点 (    k 指针 )
            pcur = firstLeaf;
        }
        else
        {
            // 链接时用叶子结点的    rchild    域存放指针
            pcur->right=root;
            pcur=pcur->right;
        }
    }
    if(root->left)
        leafLink(root->left);
    if(root->right)
        leafLink(root->right);
    return;
}
```

13 连续正整数之和

一个正整数有可能可以被表示为 $n(n \geq 2)$ 个连续正整数之和，如：

$15=1+2+3+4+5$

$15=4+5+6$

15=7+8

请编写程序，根据输入的任何一个正整数，找出符合这种要求的所有连续正整数序列。输

入数据：一个正整数，以命令行参数的形式提供给程序。

输出数据：在标准输出上打印出符合题目描述的全部正整数序列，每行一个序列，每个序列都从该序列的最小正整数开始、以从小到大的顺序打印。如果结果有多个序列，按各序列的最小正整数的大小从小到大打印各序列。此外，序列不允许重复，序列内的整数用一个空格分隔。如果没有符合要求的序列，输出“NONE”。

例如，对于 15，其输出结果是：

1 2 3 4 5

4 5 6

7 8

对于 16，其输出结果是：NONE

分析：

根据 $(a1+an)(an-a1+1)=2*m$ ，设 2 个参数 $a1, an$ ，进行 2 次循环来判定得出结果

源程序：

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{
    int m;
    int a1,an,t;
    int i=0;
    cout<<"input an integer:";
    cin>>m;
    m=2*m;
    for(an=int(sqrt(m));an<=(m/4+1);an++)
    {
        for(a1=1;a1<an;a1++)
        {
            t=(a1+an)*(an-a1+1);
            if(t==m) break;
        }
        if(a1<an)
        {
            for(i=a1;i<=an;i++)
            cout<<i<<' ';
            cout<<endl;
        }
    }
    if(i==0)
        cout<<"None";
    return 0;
}
```

```
}
```

运行结果：

```
input an integer:15
1 2 3 4 5
4 5 6
7 8
```

解法二：

用连续数的个数来做只需要 $O(n)$ ，利用 $n*(2a1-1+n)=2m$ ，经过 $2m\%n$ 和 $2a1\%2$ 判断来的出结果

源程序：

```
#include<stdio.h>
#include<math.h>
int main()
{
    int m;
    int n,a1,a;
    int i=0;
    scanf("%d",&m);
    m=2*m;
    for(n=(int)(sqrt(m));n>1;n--)
        if(m%n==0)
        {
            a=m/n-n+1;
            if(a%2==0)
            {
                a1=a/2;
                for(i=a1;i<(n+a1);i++)
                    printf("%d ",i);
                printf("\n");
            }
        }
    if(i==0)
        printf("None");
    return 0;
}
```

14 文件中有一组整数，要求排序后输出到另一个文件中

解：

```
#include<iostream.h>
#include<vector>
#include<fstream>
```

```
using namespace std;
```

```
void sort(vector<int>& data) //      冒泡排序
```

```
{
    int i,j,temp;
    int count=data.size();
    for(i=0;i<count-1;i++)
    {
        for(j=0; j<count-1-i; j++)
        {
            if(data[j]>data[j+1])
            {
                temp=data[j];
                data[j]=data[j+1];
                data[j+1]=temp;
            }
        }
    }
}
```

```
void main(void)
```

```
{
    vector<int>data;
    ifstream in("c:\\data.txt");
    if(!in)
    {
        cout<<"Can not open file data.txt!";
        exit(1);
    }
    int temp;
    while (!in.eof())
    {
        in>>temp;
        data.push_back(temp);
    }
    in.close(); //      关闭输入文件流
    sort(data);
    ofstream out("c:\\result.txt");
    if (!out)
    {
        cout<<"Can not open file result.txt!";
        exit(1);
    }
    for(int i=0;i<data.size();i++)
```

```

        out<<data[i]<<" ";
    out.close(); //    关闭输出文件流
}

```

15 小猪吃米

题目：在国际象棋的棋盘上面有 $N \times N$ 个格。每个格里面有若干的米粒。一只小猪站在 1×1 的格子里，小猪每次只能向高位的列或行移动。小猪会吃掉所经过的格子里面所有的米粒。请编写程序计算小猪能吃掉的米粒的最大值。

解：假设小猪从 $(0,0)$ 开始到棋盘上任一点 (m,n) 所能吃到的最多米粒数为 $f(m,n)$ ，则 $f(m,n)$ 满足下列关系式：

$$f(m,n) = \max\{f(m,n-1), f(m-1,n)\} + \text{Matrix}[m][n];$$

注意：

$$f(0,j) = f(0, j-1) + \text{matrix}[0][j], 0 \leq j \leq N-1$$

$$f(i,0) = f(i-1, 0) + \text{matrix}[i][0], 0 \leq i \leq N-1$$

上面分析的思路实际上是典型的动态规划思路。

源程序：

```

#include <stdio.h>
#define MAX(a, b) ((a > b) ? a : b)
int matrix[4][4] = {{2,2,3,0},
                    {0,3,1,1},
                    {1,2,2,1},
                    {4,1,2,2}};

int count[4][4];

// 初始化小猪在第 0 行或第 0 列所有位置所能吃到的最大米粒数
void initialize (void)
{
    count[0][0] = matrix[0][0];
    for (int i=1; i < 4; i++)
    {
        count[0][i] = count[0][i-1] + matrix[0][i];
        count[i][0] = count[i-1][0] + matrix[i][0];
    }
}

// 找出所能吃到的最大的米粒数
int find_max (int i, int j)
{

```

```

        if ( i == 0 )
        {
            return count[0][j];
        }
        else if ( j == 0 )
        {
            return count[i][0];
        }

        int count1 = find_max (i, j-1);
        int count2 = find_max (i-1, j);
        count[i][j] = matrix[i][j] + MAX (count1, count2);
        return count[i][j];
    }

// 打印出小猪吃米的完整路径
void print_path (int i, int j)
{
    if ( i >= 0 && j >= 0 )
    {
        if ( count[i][j] == count[i-1][j] + matrix[i][j] )
        {
            print_path (i-1, j);
        }
        else if ( count[i][j] == count[i][j-1] + matrix[i][j] )
        {
            print_path (i, j-1);
        }
        printf ("%d,%d->", i, j);
    }
}

// 打印出小猪走到矩阵中任一点所能吃到的最大米粒数
void print (void)
{
    for ( int i = 0; i < 4; ++i )
    {
        for ( int j = 0; j < 4; ++j )
        {
            printf ("%d\t", count[i][j]);
        }
        printf ("\n");
    }
}

```

```

int main (void)
{
    initialize ();

    int max = find_max (3, 3);
    printf ("count = %d\n", max);

    print ();
    printf("\nThe path is:\n");
    print_path (3, 3);
    putchar ('\n');

    return 0;
}

```

16 在一个数组中存在着新数组，求出新数组的长度。

新数组的规则为：把原数组的第 0 个元素作为新数组的第 0 个数，并把该元素的值作为下个元素的下标，再把下个元素的值作为下下个元素的下标直到碰到某个元素的值为 -1，则 -1 就是新数组的结束元素。求新数组的元素个数（包含结束元素 -1 ）。

例如：有数组 {1, 4, -1, 3, 2}，在此数组中有， A[0]=1, A[1]=4, A[4]=2, A[2]=-1。则新数组中的元素为 1, 4, 2, -1，其元素个数为 4。3 不是新数组中的元素。

提供函数的原型为 int length(int a[], int n) { }

解：

```

#include<iostream>
using namespace std;

```

```

int Length(int a[],int n)
{

```

```

    int *q = a;
    int count = 0;
    while(q)
    {
        count++;
        if(*q > n)
            break;
        int i = *q;
        if(-1 == i)
            break;
        q = a+(*q);
    }
}

```



```

    }
    return count;
}

int main(void)
{
    int array[] = {1, 4, -1, 3, 2};
    int Count = Length(array, 5);
    cout<<"The length of new array is "<<Count<<endl;
    return 0;
}

```

运行结果：

The length of new array is 4

17 写函数找出一个字符串中出现频率最高的字符（如果最高频率相同的有多个字符，取最先

遇见的那个字符）

```

#include<iostream>
using namespace std;

char Find(const char *str)
{
    int temp[255]={0};
    char retChar = 0;
    int max = 0;

    const char *p=str;
    while(*p!='\0')
    {
        temp[*p]++;
        if(temp[*p] > max)
        {
            max = temp[*p];
            retChar = *p;
        }
        // 考虑有些字符出现的频率相等的情况
        if(temp[*p] == max)
        {
            int posLast,posTemp;
            for(int i = 0; i < strlen(str); i++)

```

```

        {
            if(str[i] == retChar)
            {
                posLast = i;
                break;
            }
        }
        for(i = 0; i < strlen(str); i++)
        {
            if(str[i] == *p)
            {
                posTemp = i;
                break;
            }
        }
        if(posTemp < posLast)
            retChar = *p;
    }

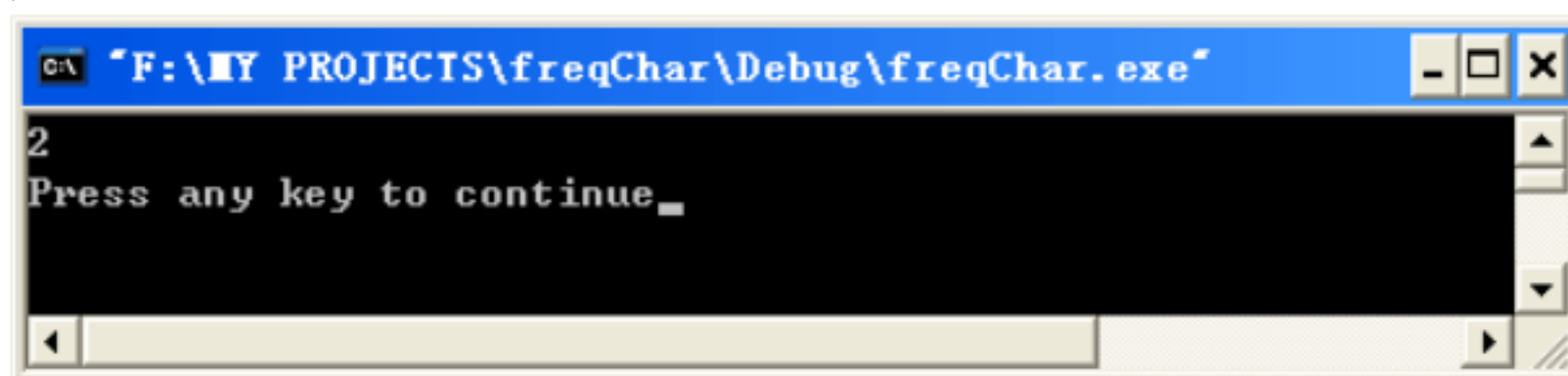
    p++;
}
return retChar;
}

```

```

void main()
{
    const char string[] = "1243334422";
    char result = Find(string);
    cout<<result<<endl;
}

```



18 十三个人围成一个圈，从第一个人开始顺序报号

1、2、3。凡是报到 “3” 者退出圈子，请

找出最后留在圈子中的人原来的序号

源程序 1：

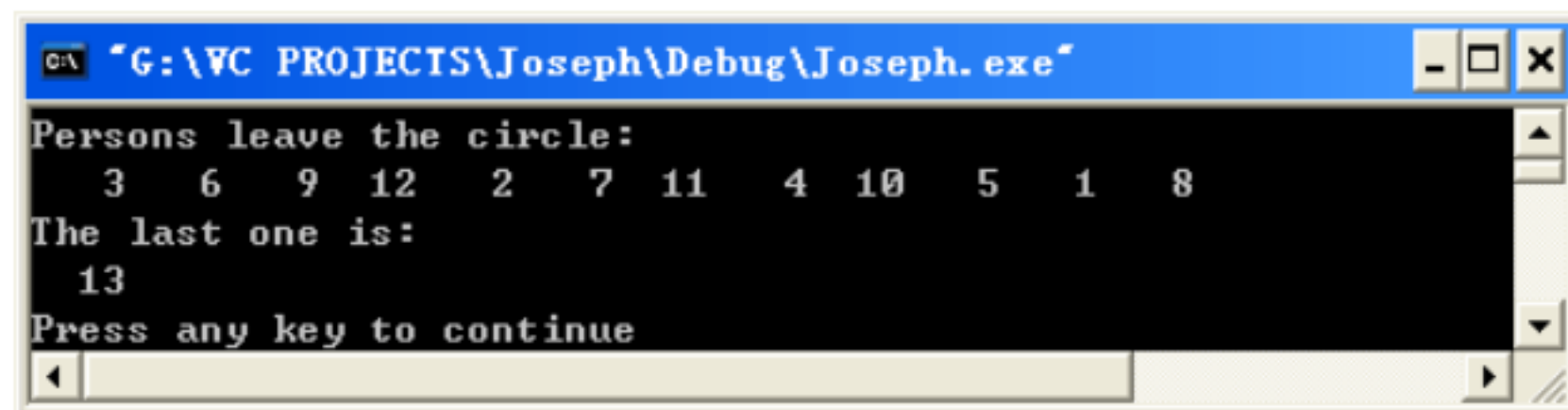
```
#include<stdio.h>
#define N 13
struct person
{
    int num;
    int next;
}link[N];

void main()
{
    int i;
    // 围成一个圈
    for(i=1;i<=N;i++)
    {
        if(i==N)
            link[i].next=1;
        else
            link[i].next=i+1;
        link[i].num=i;
    }
    // 踢人
    int count=0;
    int h=N;
    printf("Persons leave the circle: \n");
    while(count<N-1)
    {
        i=0;
        while(i!=3)
        {
            h=link[h].next;
            if(link[h].num)
                i++;
        }
        printf("%4d",link[h].num);
        link[h].num=0;
        count++;
    }
```

```

// 最后留在圈子的人
printf("\nThe last one is: \n");
for(i=1;i<=N;i++)
{
    if(link[i].num)
        printf("%4d\n",link[i].num);
}
}

```



源程序 2 :

```

#include<stdio.h>
void main()
{
    int i,n,num[50],*p;
    printf("Input number of person: n=");
    scanf("%d",&n);
    p=num;
    for(i=0;i<n;i++)
    {
        *(p+i)=i+1;          /* 以 1 至 n 为序给每个人编号 */
    }

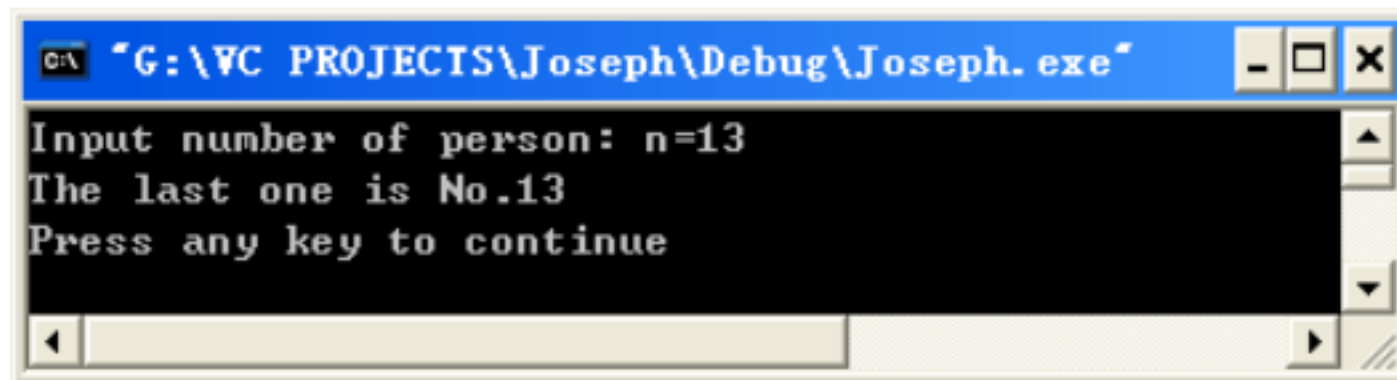
    int j=0;                  /* 按 1,2,3 报数时的计数变量 */
    int count=0;              /* 退出人数 */
    i=0;                      /* 每次循环的计数变量 */
    while(count<n-1)          /* 未退出人数大于 1 时执行循环 */
    {
        if(*(p+i)!=0) j++;
        if(j==3)              /* 对退出者编号置为 0 */
        {
            *(p+i)=0;
            j=0;
            count++;
        }
        i++;
        if(i==n) i=0;         /* 报数到最后一个后, i 恢复为 0 */
    }
}

```

```

while(*p==0)
{
    p++;
}
printf("The last one is No. %d\n", *p);
}

```



19 已知 n 个人（以编号 $1, 2, 3 \dots n$ 分别表示）围坐在一张圆桌周围。从编号为 k 的人开始报数，数到 m 的那个人出列；他的下一个人又从 1 开始报数，数到 m 的那个人又出列；依此规律重复下去，直到圆桌周围的人全部出列

解法一：

```

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

```

```

typedef struct LNode
{
    int data;
    struct LNode *next;
}LNode,*LinkList;

```

```

////////////////////////////////////
// 约瑟夫函数
// n 为总人数，k 的下一位为第一个开始报数的人，m 为出列者喊到的数
////////////////////////////////////

```

```

void JOSEPHUS(int n,int k,int m)
{
    /* pcur 为当前结点 ,pre 为辅助结点，指向 pcur 的前驱结点， head 为头节点 */
    LinkList pcur ,pre,head;
    head=NULL;
    /* 建立循环链表 */
    for(int i=1;i<=n;i++)

```

```

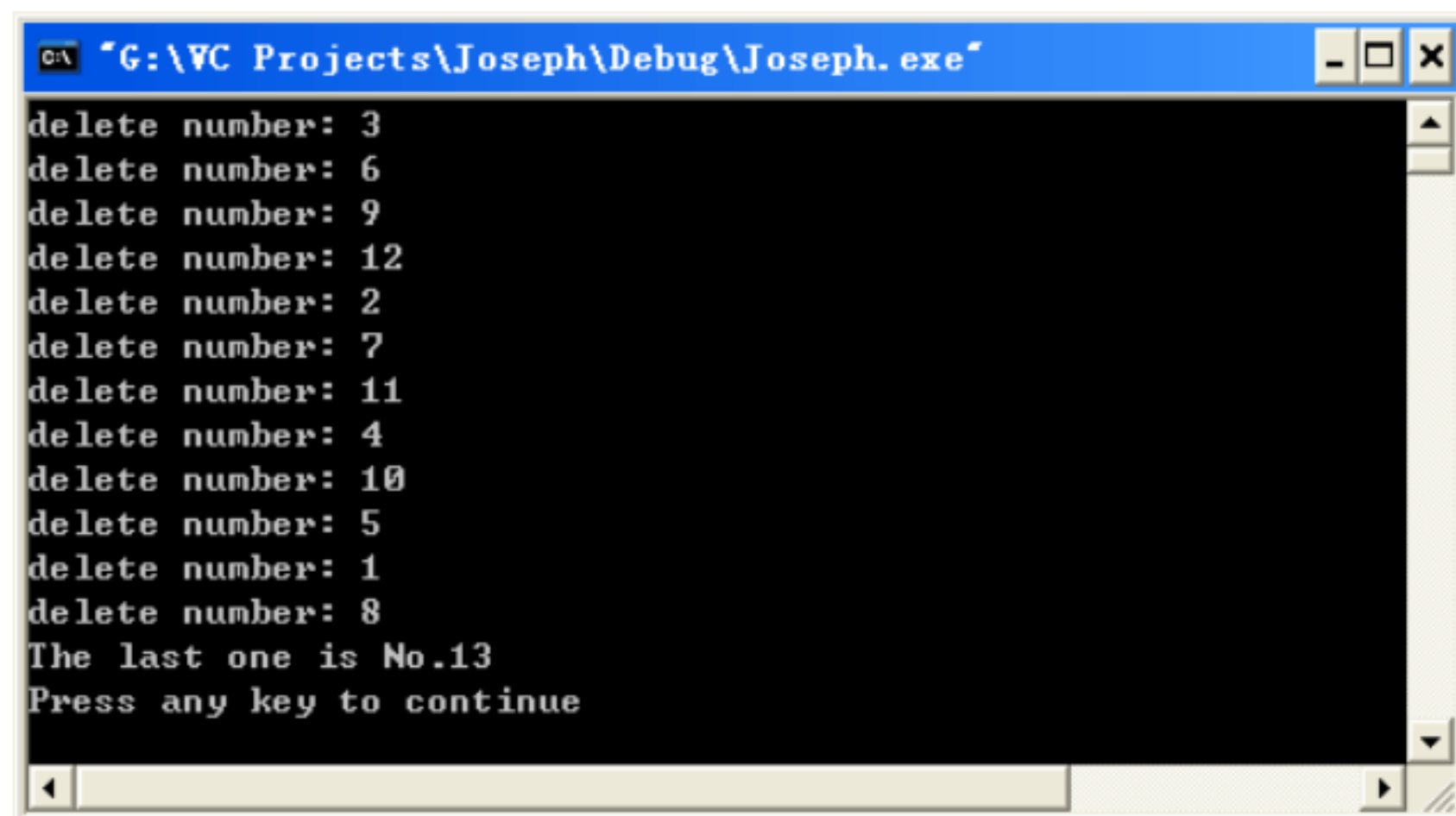
{
    pcur=(LinkedList)malloc(sizeof(LNode));
    pcur->data=i;
    if(head==NULL)
        head=pcur;
    else
        pre->next=pcur;
    pre=pcur;
}
pcur->next=head; /*      尾节点连到头结点，使整个链表循环起来      */
pcur=head; /*      使 pcur  指向头节点  */

/* 把当前指针移动到第一个报数的人，即第      k 位的下一位  */
for(i=1;i<=k;i++)
{
    pre=pcur;
    pcur=pcur->next;
}

/* 循环地删除队列结点，每隔      m-1  个结点删一个  */
while(pcur->next!=pcur)
{
    for(i=1;i<=m-1;i++)
    {
        pre=pcur;
        pcur=pcur->next;
    }
    pre->next=pcur->next;
    printf("delete number: %d\n",pcur->data);
    free(pcur);
    pcur=pre->next;
}
printf("The last one is No.%d\n",pcur->data);
}

void main()
{
    //  总共有  13  人，从第  1  位开始报数，每隔两位踢出      1  个。
    JOSEPHUS(13,13,3);
}

```



```
delete number: 3
delete number: 6
delete number: 9
delete number: 12
delete number: 2
delete number: 7
delete number: 11
delete number: 4
delete number: 10
delete number: 5
delete number: 1
delete number: 8
The last one is No.13
Press any key to continue
```

解法二：

上面的方法在时间效率上有缺陷，如果 N 和 M 的值非常大的话，时间复杂度就会非常高，我们如果换个角度来考虑这个问题的话，或许能够得到一个时间效率较高的解决方法。

n 个元素，从 0 开始，遍历到 $m-1$ 删除，剩下的元素从 0 开始，从新遍历；

在第一次遍历第一个被删除的元素一定是 $m-1\%n$ ，剩下的 $n-1$ 个元素从新组成了一个新的约瑟夫环，以编号 $k=m\%n$ 开始：K K+1 K+2 K+3...N-2 N-1 0 1 2 3...K-2

将 K 为新环的 0；上面的队列变为 0 1 2 3n-3 n-2

那么删除第一次遍历后得到节点的元素将组成一个新的约瑟夫环，遵循这一规则我们将面对的的是一个旧规则的新问题。

下面我们来实现这个递推思想：

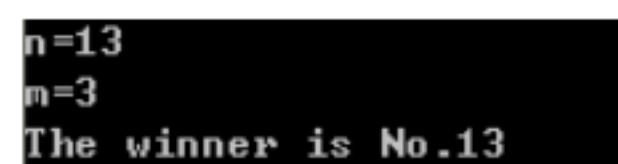
设 $a[n]$

则 $a[i]=(a[i-1]+m)\%i;(i>1)$

源程序：

```
#include <stdio.h>

void main()
{
    int n,m,i,a=0;
    printf("n=");
    scanf("%d",&n);
    printf("m=");
    scanf("%d",&m);
    for(i=2;i<=n;i++)
        a=(a+m)%i;
    printf("The winner is No.%d\n",a+1);
}
```



```
n=13
m=3
The winner is No.13
```

20 十进制正数或负数转化为二进制

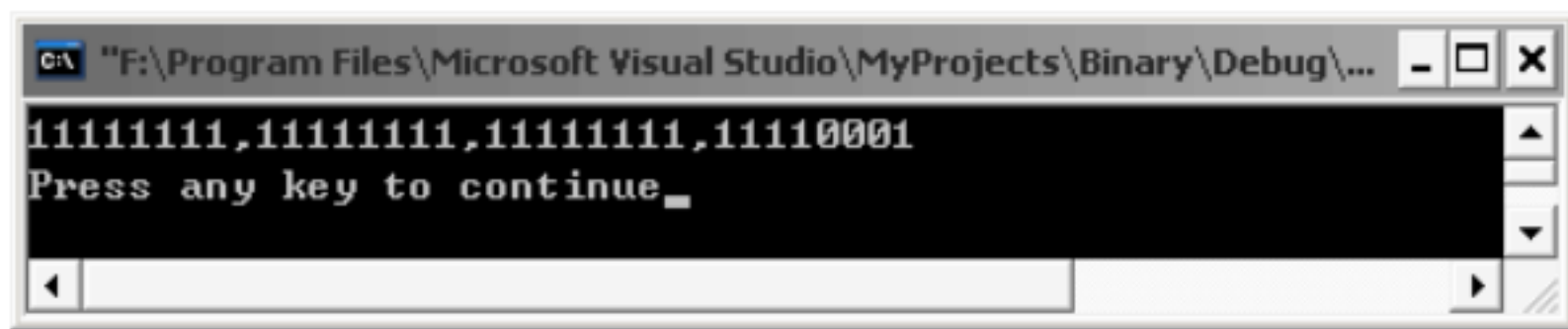
```
#include<stdio.h>

void printBinary( int x)
{
    int i;
    int j =0;
    int a[32]={0};
    // if x is an integer
    if(x>0)
    {
        while(x)
        {
            a[j++] = x % 2;
            x = x / 2 ;
        }

        // else if x is an negative integer
        else
        {
            unsigned int X = (unsigned int)x;
            while(X)
            {
                a[j++] = X % 2;
                X /= 2 ;
            }
        }

        for(i=31; i>=0; i--)
        {
            printf("%d", a[i]);
            if(i % 8 == 0 && i != 0)
                printf(",");
        }
        puts("");
    }
}

void main()
{
    printBinary(-15);
}
```

21 将阿拉伯数字转化为中文数字，如 12 “一十二”

1) 考虑不包含 0 的数字

```
#include<stdio.h>
void main()
{
    int m;
    int c[10];
    int i = 0;
    char a[10][3] = {"", "零", "一", "二", "三", "四", "五", "六", "七", "八", "九"};
    char b[10][3] = {"", "十", "百", "千", "万", "十", "百", "千", "亿", "十"};
    // int 型的最大数为 2147483647, 大于此数程序将输出错误结果
    printf("Please input an integer not larger than 21,4748,3647: ");
    scanf("%d", &m);
    while(m)
    {
        c[i++] = m % 10;
        m /= 10;
    }
    i--;
    while(i >= 0)
    {
        printf("%s%s", a[c[i]], b[i]);
        i--;
    }
    printf("\n");
}
```

2) 考虑 0 全在右边的情况，如 10,000

```
#include<stdio.h>
void main()
{
    int m;
    int c[10];
    int i = 0;
    char a[10][3] = {"", "零", "一", "二", "三", "四", "五", "六", "七", "八", "九"};
```

```

char b[10][3] = {"", "    十", " 百", " 千", " 万", " 十", " 百", " 千", " 亿", " 十"};
// int 型的最大数为 2147483647, 大于此数程序将输出错误结果
printf("Please input an integer not larger than 21,4748,3647: ");
scanf("%d", &m);
while(m)
{
    c[i++] = m % 10;
    m /= 10;
}
i--;
int zeroFlag = 0;
int flag = 0; // flag 的作用是保证 “万”只被输出一次
while(i >= 0)
{
    if(c[i] == 0)
    {
        // 考虑 “亿”和 “万”的情况
        if(i % 4 == 0 && flag == 1)
        {
            printf("%s", b[i/4*4]);
            flag = 0;
        }
        zeroFlag = 1;
    }
    else
    {
        printf("%s%s", a[c[i]],b[i]);
        if(i%4 != 0)
        {
            flag = 1;
        }
        else
        {
            flag = 0;
        }
    }
    i--;
}
printf("\n");
}

```

3) 最后考虑 0 出现在中间的情况

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

```
void main()
```

```

{
    int m;
    int c[10];
    int i = 0;
    char a[10][3] = {"零", "一", "二", "三", "四", "五", "六", "七", "八", "九"};
    char b[10][3] = {"", "十", "百", "千", "万", "十", "百", "千", "亿", "十"};
    // int 型的最大数为 2147483647, 大于此数程序将输出错误结果
    printf("Please input an integer not larger than 21,4748,3647: ");
    scanf("%d", &m);
    while(m)
    {
        c[i++] = m % 10;
        m /= 10;
    }
    i--;
    int zeroFlag = 0;
    int flag = 0; // flag 的作用是保证 “万”只被输出一次
    while(i >= 0)
    {
        if(c[i] == 0)
        {
            // 考虑 “亿”和 “万”的情况
            if(i % 4 == 0 && flag == 1)
            {
                printf("%s", b[i/4*4]);
                flag = 0;
            }
            zeroFlag = 1;
        }
        else
        {
            if(zeroFlag == 1 && i%4 != 3 )
                printf("零%s%s", a[c[i]],b[i]);
            else
                printf("%s%s", a[c[i]],b[i]);
            zeroFlag = 0;
            if(i%4 != 0)
                flag = 1;
            else
                flag = 0;
        }
        i--;
    }
    printf("\n");
}

```

```
}
```

22 大数存储，求 100 的阶乘

```
////////////////////////////////////
// 功能：    求 100 的阶乘，结果为 158 位的大数，长整形只能存 20 位数，
//          考虑用数组存储，设计数组中的每个元素可存储的最大的数为 9999
//          结果已用计算器验证正确无误
// 郑海树    2010-10-21
////////////////////////////////////

#include <stdio.h>
#define N 100    // 为了更好地理解程序，可以把 N 先设为 10 来计算

int a[64];

int main()
{
    int n,i,c,len;
    a[0]=1;
    len=1;
    for(n=N;n>1;n--)
    {
        for(c=0,i=0;i<len;i++)
        {
            long p=a[i]*n+c;
            a[i]=p%10000;
            // c 为超过 4 位的部分
            c=p/10000;
        }
        a[i]= c;
        if(c>0)
        {
            len++; // len    每加 1 结果就多了 4 位
        }
    }

    printf("%ld",a[len-1]); //          输出最高的 1~4 位，防止最高位数的左边输出 0
    for(i=len-1;i>=0; i--)
    {
        printf("%04ld",a[i]);
    }
}
```

```

    }
    printf("\n");
    return 0;
}

```

23 在一个字符串中找到第一个只出现一次的字符。如 “ abaccdeff ”, 输出 b。

```

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

int main()
{
    char c[] = "abaccdeff" ;
    int bit_map[ 26 ]={ 0};
    int i = 0;
    for (;i< strlen (c);++i)
        bit_map[c[i]- 'a' ]++;
    for (i= 0;i< strlen (c);++i)
    {
        if (bit_map[c[i]- 'a' ]== 1 )
        {
            printf ( " %c " ,c[i]);
            break ;
        }
    }
    if (i>= strlen (c))
        printf ( "No ele to the rule\n" );
    return 0 ;
}

```