大数问题、高精度计算

信智学院　陈琰宏

主要内容

01 字符转换

02 大数加

03 大数减法

04 大数乘法

05 大数除法

高精度计算

在32位机器里，有符号整数(int)的取值范围是**-2147483648 ~ +2147483647（2^{32}）**，无符号整数(unsigned int) 的取值范围是**0 ~ 4294967295**。超过这个范围的数据可以用浮点型(double)来表示，如50!。

但用浮点数来表示整数通常不便于整数的运算，比如整数除法跟浮点数除法含义不一样，浮点数也无法实现取余运算等等。

- 另外，**超过浮点数取值范围的数据**，比如一个1000位的整数，**无法用常规方法来处理**。
- 这些精度很高的数据通常称为**高精度数**，或称为**大数。**

高精度计算中需要处理好以下几个问题：

(1)数据的接收方法和存贮方法

当输入的数很长时，可采用字符串方式输入，这样可输入数字很长的数，利用字符串函数和操作运算，将每一位数取出，存入数组中。

```
1  void init(int a[])
2  {//传入一个数组,或字符串
3      char s[2000];
4      cin>>s;
5      int len=strlen(s); //用len计算字符串s的位数
6      for(i=1;i<=len;i++)
7        a[i]=s[len-i]-'0';
8        //将数串s转换为数组a并倒序存储
9  }
```

1 [3290] 输出大数的字符

输出一个大数如，
9385398593859285023589325829435359O，输出第n位的数（从右往左数）。
输入
9385398593859285023589325829435359O
10
输出
8

```
1  int n,a[200],len,i;
2  char str[200];
3  cin>>str>>n;
4  len=strlen(str);
5  for(i=1;i<=n;i++)
6     a[i]=str[len-i]-'0';
7  cout<<a[n];
```

(2) 进位,借位处理

加法进位：

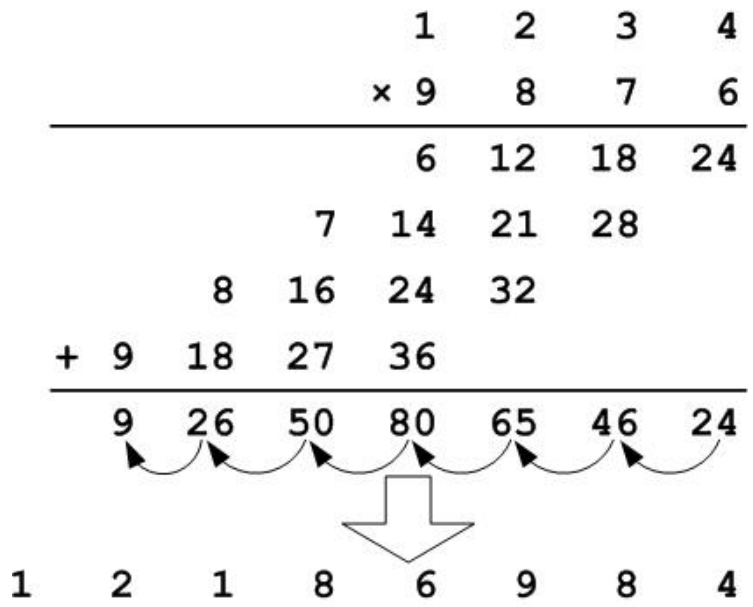
```
c[i]=a[i]+b[i];
if (c[i]>=10) {
    c[i]%=10; c[i+1]++;
 }
```

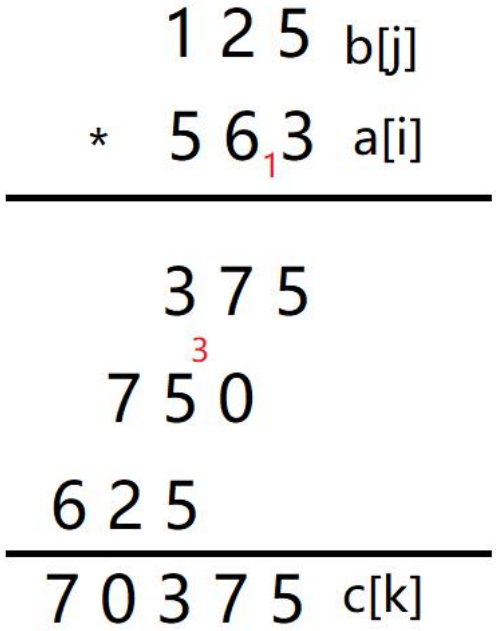
减法借位：

```
c[i]=a[i]-b[i];
if (a[i]<b[i]) {
    a[i+1]--; a[i]+=10;
}
```

(3)乘法进位：

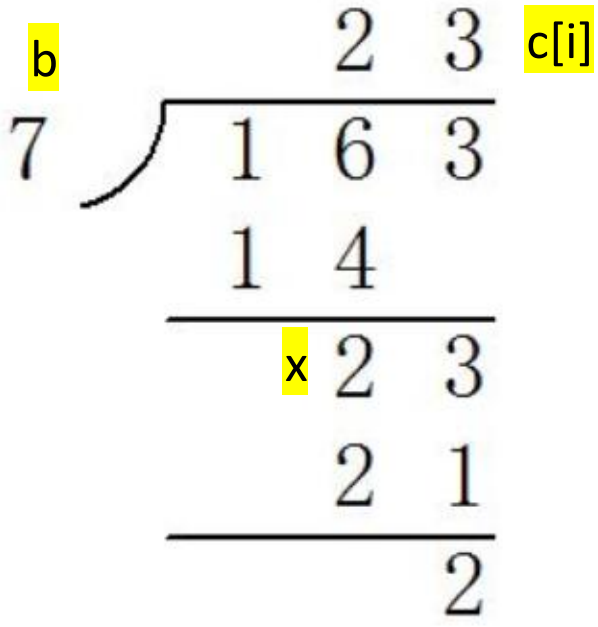
```
c[i+j-1]= a[i]*b[j] + jw + c[i+j-1];
jw = c[i+j-1]/10;
c[i+j-1] %= 10;
```





(4) 商和余数的求法

商和余数处理：视被除数和除数的位数情况进行处理.




```
x=0;
for(int i=1;i<=lena;i++){
        c[i]=(x*10+a[i])/b;
        x=(x*10+a[i])%b;
    }
```

2 [2727]高精度加法

题目描述：求两个不超过200位的非负整数的和。

输入格式：有两行，每行是一个不超过200位的非负整数，可能有多余的前导0。

输出格式：一行，即相加后的结果。结果里不能有多余的前导0，即如果结果是342，那么就不能输出为0342。

输入样例 复制
22222222222222222222
33333333333333333333
输出样例 复制
55555555555555555555

分析

在C++语言中任何数据类型都有一定的表示范围。而当两个被加数很大时，上述算法显然不能求出精确解，因此我们需要寻求另外一种方法。在读小学时，我们做加法都采用竖式方法。 这样，我们方便写出两个整数相加的算法。

$$\begin{array}{r} 8\,5\,6 \\ +\ 2\,5\,5 \\ \hline 1\,1\,1\,1 \end{array}$$

$$\begin{array}{r} A_3\,A_2\,A_1 \\ +\quad B_3\,B_2\,B_1 \\ \hline C_4\,C_3\,C_2\,C_1 \end{array}$$

分析

$$\begin{array}{r} 8\,5\,6 \\ +\ 2\,5\,5 \\ \hline 1\,1\,1\,1 \end{array} \qquad \begin{array}{r} A_3\,A_2\,A_1 \\ +\quad B_3\,B_2\,B_1 \\ \hline C_4\,C_3\,C_2\,C_1 \end{array}$$

如果我们用数组A、B分别存储加数和被加数，用数组C存储结果。则上例有A[1]=6，A[2]=5， A[3]=8，B[1]=5，B[2]=5，B[3]=2，C[4]=1，C[3]=1，C[2]=1，C[1]=1，两数相加如图所示。

算法描述：

```
1   int c[100];
2   void add(int a[],int b[])
3   {//a,b,c都为数组，分别存储被加数、加数、结果
4       int  i=1,x=0; //x是进位
5       while ((i<=a数组长度)||(i<=b数组的长度))
6     {
7           c[i]=a[i]+b[i]+x; //第i位相加并加上次的进位
8           x=c[i]/10;//向高位进位
9           c[i]%=10; //存储第i位的值
10          i++;      //位置下标变量
11    }
12  }
```

代码实现1：

```
#include<bits/stdc++.h>
using namespace std;
char str1[200],str2[200];
int a[200],b[200],c[200];
int main()
{
    cin>>str1;
    cin>>str2;
    int lena,lenb,lenc;
    lena=strlen(str1);
    lenb=strlen(str2);
    for(int i=0;i<lena;i++)
        a[i]=str1[lena-1-i]-'0';
    for(int i=0;i<lenb;i++)
        b[i]=str2[lenb-1-i]-'0';
```

代码实现2：

```
16      int jw=0;
17      int k=0;
18      while(k<lena||k<lenb)
19       {
20          c[k]=a[k]+b[k]+jw;
21          jw=c[k]/10;
22          c[k]=c[k]%10;
23          k++;
24       }
25      c[k]=jw;
26      while(c[k]==0)
27          k--;
28      for(int i=k;i>=0;i--)
29          cout<<c[i];
```

string串

string是C++标准库的一个重要的部分，主要用于字符串处理。可以使用输入输出流方式直接进行string操作，也可以通过文件等手段进行string操作。同时，C++的算法库对string类也有着很好的支持，并且string类还和c语言的字符串之间有着良好的接口。

一、初始化

初始化有两种方式，其中使用等号的是拷贝初始化，不使用等号的是直接初始化。

```
string str1 = "hello world";// str1 = "hello world"
string str3 = str1;      // str3 = "hello world"
```

二、获取长度（length、size）

```
string str = "hello world";
cout << str.length() << str.size();
```

```
string str = "hello world";
```

三、插入（insert）

```
 str.insert(pos,n,ch) //在字符串s的pos位置上面插入n个字符ch
str.insert(6,4,'z');  // str = "hello zzzzworld"
str.insert(pos,str)    // 在字符串s的pos位置插入字符串str
str.insert(6,str2);  // str = "hello hard world"
```

四、赋值（assign）

赋值也是一种初始化方法，与插入、替换、添加对应理解较为简单。

```
str.assign(n,ch)      将n个ch字符赋值给字符串s
str.assign(10,'h');   // str = "hhhhhhhhhh"
str.assign(s)    // 将字符串s赋值给字符串str
str.assign(temp);  // str = "welcome to my blog"
```

五、删除（erase）

```
string str = "welcome to my blog";
s.erase(pos,n)    把字符串s从pos开始的n个字符删除
str.erase(11,3);// str = "welcome to blog"
```

六、取子串（substr）

```
string str = "The apple thinks apple is delicious";
s.substr(pos,n)   得到字符串s位置为pos后面的n个字符组成的串
string s1 = str.substr(4,5); // s1 = "apple"
s.substr(pos)     得到字符串s从pos到结尾的串
string s2 = str.substr(17);  // s2 = "apple is delicious"
```

七、比较（compare）

两个字符串自左向右逐个字符相比（按ASCII值大小相比较），直到出现不同的字符或遇’\0’为止。

若是遇到‘\0’结束比较，则长的子串大于短的子串，如：“9856” > “985”。如果两个字符串相等，那么返回0，调用对象大于参数返回1，小于返回-1。

```
string str1 = "small leaf";
string str2 = "big leaf";
```

s.compare(str)　比较当前字符串s和str的大小

```
cout << str1.compare(str2);     // 1
```

s.compare(pos,n,str)　比较当前字符串s从pos开始的n个字符与str的大小

```
cout << str1.compare(2,7,str2);// -1
```

s.compare(pos,n0,str,pos2,n)比较当前字符串s从pos开始的n0个字符与str中pos2开始的n个字符组成的字符串的大小

```
cout << str1.compare(6,4,str2,4,4);                    // 0
```

八、交换（swap）交换两个字符串的值。

```
string str1 = "small leaf";
string str2 = "big leaf";
str1.swap(str2)  ,输出结果相同
swap(str1,str2);  // str1 = "big leaf"     str2 = "small leaf"
swap(str1[0],str1[1]);  // str1 = "ibg leaf"
```

九、反转（reverse） 反转字符串。

```
string str = "abcdefghijklmn";
reverse(str.begin(),str.end()); //str = "nmlkjihgfedcba"
```

十、 find函数

```
string str = "The apple thinks apple is delicious";
//长度34
string key = "apple";
```

s.find(str) 查找字符串str在当前字符串s中第一次出现的位置

```
int pos1 = str.find(key);          // 4
```

s.find(str,pos) 查找字符串str在当前字符串s的[pos,end]中第一次出现的位置

```
int pos2 = str.find(key, 10);    // 17
```

/从pos开始从后向前查找字符串s中前n个字符组成的字符串在当前串中的位置，成功返回所在位置，失败时返回string::npos的值

```
//从pos开始查找当前串中第一个在s的前n个字符组成的数组里的字符的位置。查找失败返回string::npos
int find_first_not_of(char c, int pos = 0) const;
int find_first_not_of(const char *s, int pos = 0) const;
int find_first_not_of(const char *s, int pos,int n) const;
int find_first_not_of(const string &s,int pos = 0) const;
//从当前串中查找第一个不在串s中的字符出现的位置，失败返回string::npos
int find_last_of(char c, int pos = npos) const;
int find_last_of(const char *s, int pos = npos) const;
int find_last_of(const char *s, int pos, int n = npos) const;
int find_last_of(const string &s,int pos = npos) const;
......
//find_last_of和find_last_not_of与find_first_of和find_first_not_of相似，只不过是从后向前查找
```

代码实现1：

```
 7  string a,b,ans;
 8  cin>>a>>b;
 9  int lena=a.length();//求长度
10  int lenb=b.length();
11  int mi=min(lena,lenb);
12  int ma=max(lena,lenb);
13  reverse(a.begin(), a.end());//倒过来
14  reverse(b.begin(), b.end());
```

```
15  for(int i=0;i<mi;i++)
16      ans+=a[i]+b[i]-'0';
17  if (lena<lenb)
18  {
19      for (int i =mi; i <ma; i++)
20          ans += b[i];
21  }
22  else if (lena >lenb) {
23      for (int i =mi; i <ma; i++)
24          ans += a[i];
25  }
```

代码实现2：

```
27  ans += '0';
28  for (int i = 0; i <ma + 1; i++) {
29      if (ans[i] > '9') {
30          ans[i + 1]++;
31          ans[i] -= 10;
32      }
33  }
34  reverse(ans.begin(), ans.end());
35  for (int i = ans.find_first_not_of('0'); i <ma + 1; i++) {
36      if (i == -1) {
37          cout << 0;
38          return 0;
39      }
40      cout << ans[i];
41  }
```

3 [2728]高精度减法

题目描述
求两个大的正整数相减的差。

输入格式
共2行，第1行是被减数a，第2行是减数b(a ＞ b)。每个大整数不超过200位，不会有多余的前导零。

输出格式
一行，即所求的差。

分析

类似加法，可以用竖式求减法。在做减法运算时，需要注意的是：被减数必须比减数大，同时需要处理借位。

$$\begin{array}{r} 1\,0\,0\,0 \\ -\quad 3\,3\,7 \\ \hline 6\,6\,3 \end{array}$$

数据输入：

```
7   int a[256],b[256],c[256],lena,lenb,lenc,i;
8   char n[256],n1[256],n2[256];
9   memset(a,0,sizeof(a));
10  memset(b,0,sizeof(b));
11  memset(c,0,sizeof(c));
12  gets(n1);    //输入被减数
13  gets(n2);    //输入减数
14  if (strlen(n1)<strlen(n2)||(strlen(n1)==strlen(n2)&&strcmp(n1,n2)<0))
15  //strcmp()为字符串比较函数，当n1==n2，返回0;
16  //n1>n2时，返回正整数；n1<n2时，返回负整数
17  {  //处理被减数和减数，交换被减数和减数
18      strcpy(n,n1);//将n1数组的值完全赋值给n数组
19      strcpy(n1,n2);
20      strcpy(n2,n);
21      cout<<"-";  //交换了减数和被减数，结果为负数
22  }
```

相减运算：

```
23  lena=strlen(n1); lenb=strlen(n2);
24  for (i=0;i<=lena-1;i++)
25      a[lena-i]=int(n1[i]-'0'); //被减数放入a数组
26  for (i=0;i<=lenb-1;i++)
27      b[lenb-i]=int(n2[i]-'0'); //减数放入b数组
28  i=1;
29  while (i<=lena||i<=lenb)
30  {
31      if (a[i]<b[i])
32      {
33          a[i]+=10; //不够减，那么向高位借1当10
34          a[i+1]--;
35      }
36      c[i]=a[i]-b[i];//对应位相减
37      i++;
38  }
```

输出结果：

```
40  lenc=i;
41  while ((c[lenc]==0)&&(lenc>1))
42      lenc--;    //最高位的0不输出
43  for (i=lenc;i>=1;i--)
44      cout<<c[i];//输出结果
45  cout<<endl;
```

[2733]高精度乘法

题目描述

求两个不超过200位的非负整数的积。

输入格式

有两行，每行是一个不超过200位的非负整数，没有多余的前导0。

输出格式

一行，即相乘后的结果。结果里不能有多余的前导0，即如果结果是342，那么就不能输出为0342。

算法分析

类似加法，可以用竖式求乘法。在做乘法运算时，同样也有进位，同时对每一位进行乘法运算时，必须进行错位相加。

分析c数组下标的变化规律，可以写出如下关系式：

$$c_i = c'_i + c''_i + \ldots$$

由此可见，c_i跟a[i]*b[j]乘积有关，跟上次的进位有关，还跟原c_i的值有关，分析下标规律，有

```
c[i+j-1]= a[i]*b[j]+ jw + c[i+j-1];
x=c[i+j-1]/10 ;
c[i+j-1]%=10;
```

$$\begin{array}{r} 856 \\ \times\ 25 \\ \hline 4280 \\ 1712 \\ \hline 21400 \end{array}$$

$$\begin{array}{r} A_3A_2A_1 \\ \times\ B_2B_1 \\ \hline C'_4C'_3C'_2C'_1 \\ C''_5C''_4C''_3C''_2 \\ \hline C_6C_5C_4C_3C_2C_1 \end{array}$$

算法分析

```
c[i+j-1]= a[i]*b[j] + jw + c[i+j-1];
jw = c[i+j-1]/10;
c[i+j-1] %= 10;
```

```
    1 2 5   b[j]
  *   5 6 3   a[i]
-------------
      3 7 5
    7 5 0
  6 2 5
-------------
  7 0 3 7 5   c[k]
```

代码实现1：

```
 6  char a1[205],b1[205];
 7  int a[205],b[205],c[205];
 8  int lena,lenb,lenc,i,x;
 9  memset(a,0,sizeof(a));
10  memset(b,0,sizeof(b));
11  memset(c,0,sizeof(c));
12  cin>>a1;
13  cin>>b1;
14  lena=strlen(a1);
15  lenb=strlen(b1);
16  for(i=0;i<=lena-1;i++)a[lena-i]=a1[i]-48;
17  for(i=0;i<=lenb-1;i++)b[lenb-i]=b1[i]-48;
```

代码实现2：

```
18  for(i=1;i<=lena;i++){//a[i]
19      x=0;//进位
20      for(int j=1;j<=lenb;j++){//b[j]
21          c[i+j-1]=a[i]*b[j]+x+c[i+j-1];
22          x=c[i+j-1]/10;//生成进位
23          c[i+j-1]=c[i+j-1]%10;//c[k]
24      }
25      c[i+lenb]=x;//存储最高位进位
26  }
27  lenc=lena+lenb;
28  while(c[lenc]==0&&lenc>1)
29      lenc--;//去掉高位0
30  for(i=lenc;i>=1;i--)
31      cout<<c[i];
32  cout<<endl;
```

[2734] 除以13

输入一个大于0的大整数N，长度不超过4000位，要求输出其除以13得到的商和余数。
输入格式：一个大于0的大整数，长度不超过4000位。

输出格式：两行，分别为整数除法得到的商和余数。

输入样例
2132104848488485
输出样例
164008065268345
0

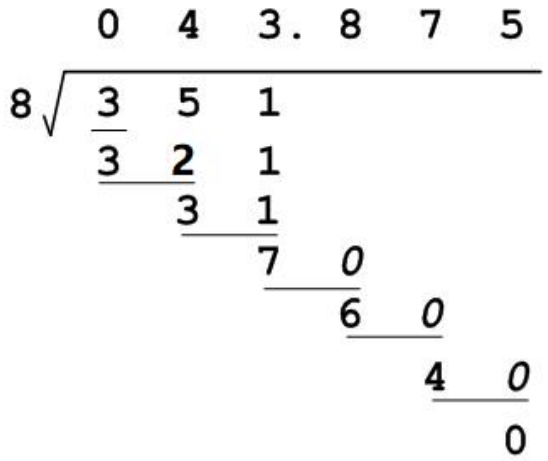
分析

大数除法，是四则运算里面最难的一种。不同于一般的模拟，除法操作步数模仿手工除法，而是利用减法操作实现的。如图所示，为普通除法处理过程。

1）先将被除数"351"最高位除以8，得到的商为0，余数为3。

2）把余数和被除数"351"的次高位组合，为"35"，除以8，得到的商为4，余数为3。

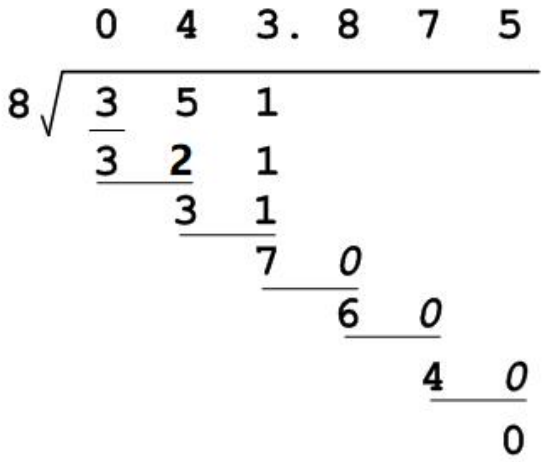
3）把余数和被除数"351"的最后一位组合，为"31"，除以8，得到的商为3，余数为7，不为0，所以还没有除尽，要补0，图中所有补0均用斜体标明。补0前的商为整数部分，补0后的商为小数部分。



除法运算过程(除数只有1位数)

分析

4）补0后为"70"，除以8，得到的商为8，余数为6，再补0。
5）补0后为"60"，除以8，得到的商为7，余数为4，再补0。
6）补0后为"40"，除以8，得到的商为5，余数为0。
至此，整个除法运算完毕，得到的商为43.875

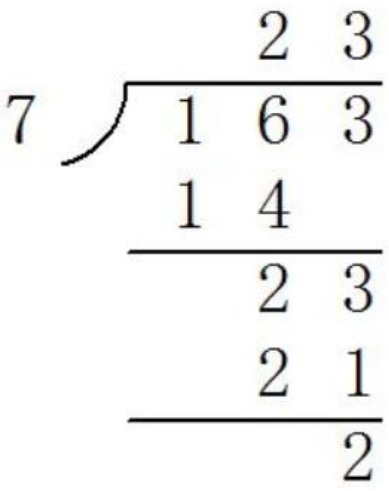


除法运算过程(除数只有1位数)

分析

做除法时，每一次上商的值都在 0 ～ 9 ，每次求得的余数连接以后的若干位得到新的被除数，继续做除法。因此，在做高精度除法时，要涉及到乘法运算和减法运算，还有移位处理。当然，为了程序简洁，可以避免高精度除法，用0～9次循环减法取代得到商的值。这里，我们采取的方法是**按位相除法**。

```
for(int i=1;i<=lena;i++){
    c[i]=(x*10+a[i])/b;//模拟除法
    x=(x*10+a[i])%b;   //b=13
}
```



代码实现

```
 5  char str[4000];
 6  int a[4000],c[4000],lena,lenc,b=13,x=0;
 7  memset(a,0,sizeof(a));
 8  memset(c,0,sizeof(c));
 9  cin>>str;
10  lena=strlen(str);
11  for(int i=0;i<lena;i++){
12      a[i+1]=str[i]-'0';//
13  }//把输入的字符存储为数字
```

代码实现

```
15  //如 5321/13, 先5除以13余数为5,c[i]=0;,c[i]存储商
16  //的每一位 接着余数5连接上下一位，得到53，53除以13，
17  //得到当前的c[i],和新余数x。
18  for(int i=1;i<=lena;i++){
19      c[i]=(x*10+a[i])/b;//模拟除法
20      x=(x*10+a[i])%b;   //b=13
21  }
22  lenc=1;
23  while(c[lenc]==0&&lenc<lena)lenc++;
24  for(int i=lenc;i<=lena;i++ )cout<<c[i];
25  cout<<endl;
26  cout<<x;
```

1049

[3032] 回文数

若一个数（首位不为零）从左向右读与从右向左读都是一样，我们就将其称之为回文数。例如：给定一个 10进制数56，将 56加 65（即把56从右向左读），得到 121是一个回文数。又如，对于10进制数87，

STEP1：87＋78= 165

STEP2：165＋561= 726

STEP3：726＋627＝1353

STEP4：1353+3531=4884

在这里的一步是指进行了一次N进制的加法，上例最少用了4步得到回文数4884。

写一个程序，给定一个N（2＜N＜＝10或N=16）进制数M．求最少经过几步可以得到回文数。如果在30步以内（包含30步）不可能得到回文数，则输出“Impossible”

算法分析

按照题目要求进行模拟即可，重点在于：

1. 判断一个数是否是回文数
2. 因为题目没有说数据的范围，按照大数相加处理

代码实现

```
38  int main()
39  {
40      init(a);//将数串s转化为整数数组a
41      if(check(a)){//判别原数是否为回文数
42          cout<<0<<endl;
43          return 0;
44      }
45      ans=0;  //步数初始化为0
46      while(ans++<=30)
47      {
48          jia(a);//整数数组a与其反序数b进行n进制加法运算
49          if(check(a)){
50              cout<<"STEP="<<ans<<endl;
51              return 0;
52          }
53      }
54      cout<<"Impossible!";    //输出无解信息
55      return 0;
56  }
```

代码实现

```
 4  int n,a[101],b[101],i;
 5  int ans,len; //ans表示步数，len表示长度
 6  void init(int a[])
 7  { //将数串s转化为整数数组a
 8      string s;
 9      cin>>n>>s; //读入字符串s
10      memset(a,0,sizeof(a)); //数组a清0
11      len=s.length();    //用len计算字符串s的位数
12      for(i=1;i<=len;i++){
13          if(s[len-i]>='0'&&s[len-i]<='9')
14              a[i]=s[len-i]-'0';
15          else
16              a[i]=s[len-i]-'A'+10;//如果十六进制就有A~F
17      }
18  }
```

代码实现

```
19  bool check(int a[])
20  { //判别整数数组a是否为回文数
21      for(i=1;i<=len;i++)
22          if(a[i]!=a[len-i+1])return false;
23      return true;
24  }
25  void  jia(int a[])
26  {//整数数组a与其反序数b进行n进制加法运算
27      for(int i=1;i<=len;i++)
28          b[i]=a[len-i+1]; //反序数b
29      for(int i=1;i<=len;i++)
30          a[i]+=b[i];  //逐位相加
31      for(int i=1;i<=len;i++) //处理进位
32      {   a[i+1]+=a[i]/n;
33          a[i]%=n;
34      }
35      if(a[len+1]>0) len++;
36      //修正新的a的位数（a+b最多只能的一个进位）
37  }
```