第一章 高精度计算

利用计算机进行数值计算，有时会遇到这样的问题：有些计算要求精度高，希望计算的数的位数可达几十位甚至几百位，虽然计算机的计算精度也算较高了，但因受到硬件的限制，往往达不到实际问题所要求的精度。我们可以利用程序设计的方法去实现这样的高精度计算。介绍常用的几种高精度计算的方法。

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

(2) 高精度数位数的确定

位数的确定：接收时往往是用字符串的，所以它的位数就等于字符串的长度。

(3) 进位,借位处理

(4) 商和余数的求法

(1)数据的接收方法和存贮方法

数据的接收和存贮：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。另一种方法是直接用循环加数组方法输入数据。

```
void init(int a[])                          //传入一个数组
{
    string s;
    cin>>s;                                 //读入字符串s
    a[0]=s.length();                        //用a[0]计算字符串s的位数
    for(i=1;i<=a[0];i++)
        a[i]=s[a[0]-i]-'0';     //将数串s转换为数组a，并倒序存储
}另一种方法是直接用循环加数组方法输入数据。
```

(2) 高精度数位数的确定

位数的确定：接收时往往是用字符串的，所以它的位数就等于字符串的长度。

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

c[i]=a[i]-b[i];

乘法进位：c[i+j-1]= a[i]*b[j] + x + c[i+j-1];

x = 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.

【例1】高精度加法。输入两个正整数，求它们的和。

【分析】

输入两个数到两个变量中，然后用赋值语句求它们的和，输出。但是，我们知道，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，如图1。 这样，我们方便写出两个整数相加的算法。

```
  8 5 6
+ 2 5 5
---------
  1 1 1 1
```

图1

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

图2

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2所示。

因此，算法描述如下：

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

通常，读入的两个整数用可用字符串来存储，程序设计如下：

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
int main()
{
    char a1[100],b1[100];
    int a[100],b[100],c[100],lena,lenb,lenc,i,x;
    memset(a,0,sizeof(a));
    memset(b,0,sizeof(b));
    memset(c,0,sizeof(c));
    gets(a1);
    gets(b1);                                   //输入加数与被加数
    lena=strlen(a1);
    lenb=strlen(b1);
    for (i=0;i<=lena-1;i++) a[lena-i]=a1[i]-48;    //加数放入a数组
    for (i=0;i<=lenb-1;i++) b[lenb-i]=b1[i]-48;   //加数放入b数组
    lenc =1;
    x=0;
```

```
    while (lenc <=lena||lenc <=lenb)
    {
            c[lenc]=a[lenc]+b[lenc]+x;     //两数相加
            x=c[lenc]/10;
            c[lenc]%=10;
            lenc++;
    }
    c[lenc]=x;
    if (c[lenc]==0)
       lenc--;                              //处理最高进位
    for (i=lenc;i>=1;i--)
    cout<<c[i];                                 //输出结果
    cout<<endl;
    return 0;
}
```

【例2】高精度减法。输入两个正整数，求它们的差。

【算法分析】

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。高精度减法的参考程序：

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
int main()
{
    int a[256],b[256],c[256],lena,lenb,lenc,i;
    char n[256],n1[256],n2[256];
    memset(a,0,sizeof(a));
    memset(b,0,sizeof(b));
    memset(c,0,sizeof(c));
```

```
 printf("Input minuend:");    gets(n1);   //输入被减数
 printf("Input subtrahend:"); gets(n2);   //输入减数
if (strlen(n1)<strlen(n2)||(strlen(n1)==strlen(n2)&&strcmp(n1,n2)<0))
                        //strcmp()为字符串比较函数，当n1==n2, 返回0;
                       //n1>n2时，返回正整数；n1<n2时，返回负整数
{                                  //处理被减数和减数，交换被减数和减数
        strcpy(n,n1);              //将n1数组的值完全赋值给n数组
        strcpy(n1,n2);
        strcpy(n2,n);
        cout<<"-";                 //交换了减数和被减数，结果为负数
}

lena=strlen(n1); lenb=strlen(n2);
for (i=0;i<=lena-1;i++) a[lena-i]=int(n1[i]-'0');  //被减数放入a数组
for (i=0;i<=lenb-1;i++) b[lenb-i]=int(n2[i]-'0');  //减数放入b数组
```

```
    i=1;
    while (i<=lena||i<=lenb)
    {
       if (a[i]<b[i])
       {
            a[i]+=10;              //不够减，那么向高位借1当10
            a[i+1]--;
       }
       c[i]=a[i]-b[i];                    //对应位相减
       i++;
    }
    lenc=i;
    while ((c[lenc]==0)&&(lenc>1)) lenc--;   //最高位的0不输出
    for (i=lenc;i>=1;i--) cout<<c[i];              //输出结果
    cout<<endl;
    return 0;
}
```

【例3】高精度乘法。输入两个正整数，求它们的积。

【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，如图3、图4。

分析c数组下标的变化规律，可以写出如下关系式：

$C_i = C'_i + C''_i + \cdots$由此可见，c ¡跟a[i]*b[j]乘积有关，跟上次的进位有关，还跟原c ¡的值有关，分析下标规律，有c[i+j-1]= a[i]*b[j]+ x + c[i+j-1]; x=c[i+j-1]/10 ; c[i+j-1]%=10;

```
   8 5 6
 × 2 5
-------
   4 2 8 0
 1 7 1 2
-------
 2 1 4 0 0
```

图3

$$\begin{array}{r} A_3\ A_2\ A_1 \\ \times\ B_2\ B_1 \\ \hline C'_4\ C'_3\ C'_2\ C'_1 \\ C''_5\ C''_4\ C''_3\ C''_2 \\ \hline C_5\ C_4\ C_3\ C_2\ C_1 \end{array}$$

图4

高精度乘法的参考程序

```
1.  #include<iostream>
2.  #include<cstring>
3.  #include<cstdio>
4.  using namespace std;
5.  int main()
6.  {
7.      char a1[100],b1[100];
8.      int a[100],b[100],c[100],lena,lenb,lenc,i,j,x;
9.      memset(a,0,sizeof(a));
10.     memset(b,0,sizeof(b));
11.     memset(c,0,sizeof(c));
12.     gets(a1);gets(b1);
13.     lena=strlen(a1);lenb=strlen(b1);
14.     for (i=0;i<=lena-1;i++) a[lena-i]=a1[i]-48;
15.     for (i=0;i<=lenb-1;i++) b[lenb-i]=b1[i]-48;
16.     for (i=1;i<=lena;i++)
17.     {
18.         x=0;                    //用于存放进位
19.         for (j=1;j<=lenb;j++)
          //对乘数的每一位进行处理
20.         {
21.             c[i+j-1]=a[i]*b[j]+x+c[i+j-1];
              //当前乘积+上次乘积进位+原数
22.             x=c[i+j-1]/10;
23.             c[i+j-1] %= 10;
24.         }
25.         c[i+lenb]=x;                    //进位
26.     }
27.     lenc=lena+lenb;
28.     while (c[lenc]==0&&lenc>1)    //删除前导0
29.         lenc--;
30.         for (i=lenc;i>=1;i--)
31.                     cout<<c[i];
32.         cout<<endl;
33.         return 0;
34. }
```

【例4】高精度除法。输入两个正整数，求它们的商（做整除）。

【算法分析】

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，我们讨论一下高精度数除以单精度数的结果，采取的方法是按位相除法。

整除代码

```
1.  #include<iostream>
2.  #include<cstring>
3.  #include<cstdio>
4.  using namespace std;
5.  int main()
6.  {
7.    char a1[100],c1[100];
8.    int a[100],c[100],lena,i,x=0,lenc,b;
9.    memset(a,0,sizeof(a));
10.   memset(c,0,sizeof(c));
11.   gets(a1);
12.   cin>>b;
13.   lena=strlen(a1);
14.   for (i=0;i<=lena-1;i++)
15.       a[i+1]=a1[i]-48;
```

```
16.     for (i=1;i<=lena;i++)      //按位相除
17.     {
18.       c[i]=(x*10+a[i])/b;
19.       x=(x*10+a[i])%b;
20.     }
21.     lenc=1;
22.     while (c[lenc]==0&&lenc<lena)
23.       lenc++;                  //删除前导0
24.     for (i=lenc;i<=lena;i++)
25.       cout<<c[i];
26.     cout<<endl;
27.     return 0;
28. }
```

实质上，在做两个高精度数运算时候，存储高精度数的数组元素可以不仅仅只保留一个数字，而采取保留多位数（例如一个整型或长整型数据等），这样，在做运算（特别是乘法运算）时，可以减少很多操作次数。例如图5就是采用4位保存的除法运算，其他运算也类似。具体程序可以修改上述例题予以解决，程序请读者完成。

示例：123456789 ÷45 = 1’ 2345’ 6789 ÷ 45
= 274’ 3484
∵ 1 / 45 = 0 ， 1%45=1
∴ 取12345 / 45 = 274 ∵ 12345 % 45 = 15
∴ 取156789/45 = 3484
∴ 答案为2743484, 余数为156789%45 = 9

图5

【例5】高精除以高精，求它们的商和余数。

【算法分析】

高精除以低精是对被除数的每一位（这里的“一位”包含前面的余数，以下都是如此）都除以除数，而高精除以高精则是用减法模拟除法，对被除数的每一位都减去除数，一直减到当前位置的数字（包含前面的余数）小于除数（由于每一位的数字小于10，所以对于每一位最多进行10次计算）具体实现程序如下：

```
#include<iostream>
#include<cstring>
using namespace std;
int a[101],b[101],c[101],d,i;
void init(int a[])
{    string s;
     cin>>s;                    //读入字符串s
     a[0]=s.length();           //用a[0]计算字符串 s的位数
     for(i=1;i<=a[0];i++)
     a[i]=s[a[0]-i]-'0';        //将数串s转换为数组a，并倒序存储.
}
void print(int a[])             //打印输出
{
     if (a[0]==0){cout<<0<<endl;return;}
     for(int i=a[0];i>0;i--) cout<<a[i];
     cout<<endl;
     return ;
}
```

```
int compare (int a[],int b[])
                //比较a和b的大小关系，若a>b则为1，a<b则为-1,a=b则为0
{   int i;
    if(a[0]>b[0]) return 1;                //a的位数大于b则a比b大
    if(a[0]<b[0]) return -1;               //a的位数小于b则a比b小
    for(i=a[0];i>0;i--)                    //从高位到低位比较
    {
        if (a[i]>b[i]) return 1;
        if (a[i]<b[i]) return -1;
    }
    return 0;                              //各位都相等则两数相等。
}

void numcpy(int p[],int q[],int det)     //复制p数组到q数组从det开始的地方
{
    for (int i=1;i<=p[0];i++) q[i+det-1]=p[i];
    q[0]=p[0]+det-1;
}
```

```
void jian(int a[],int b[])              //计算a=a-b
{
    int flag,i;
    flag=compare(a,b);                  //调用比较函数判断大小
    if (flag==0) {a[0]=0;return;}   //相等
    if(flag==1)                         //大于
    {
       for(i=1;i<=a[0];i++)
       {
            if(a[i]<b[i]){ a[i+1]--;a[i]+=10;}        //若不够减则向上借一位
            a[i]-=b[i];
       }
       while(a[0]>0&&a[a[0]]==0) a[0]--;              //修正a的位数
       return;
    }
}
```

```
void chugao(int a[],int b[],int c[])
{
    int tmp[101];
    c[0]=a[0]-b[0]+1;
    for (int i=c[0];i>0;i--)
    {
        memset(tmp,0,sizeof(tmp));                    //数组清零
        numcpy(b,tmp,i);
        while(compare(a,tmp)>=0){c[i]++;jian(a,tmp);}  //用减法来模拟
    }
    while(c[0]>0&&c[c[0]]==0)c[0]--;
    return ;
}
```

```
int main()
{
    memset(a,0,sizeof(a));
    memset(b,0,sizeof(b));
    memset(c,0,sizeof(c));
    init(a);init(b);
    chugao(a,b,c);
    print(c);
    print(a);
    return 0;
}
```

【例6】回文数

【问题描述】

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，

STEPl： 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”

【输入样例】：9 87

【输出样例】：6

【算法分析】

N进制运算

1、当前位规范由%10改为% n

2、进位处理由/10改为/n

3、其他运算规则不变

【参考程序】

```
#include<iostream>
#include<cstring>
using namespace std;
int n,a[101],b[101],ans,i;
void init(int a[])                          //将数串s转化为整数数组a
{
   string s;
   cin>>n>>s;                               //读入字符串s
   memset(a,0,sizeof(a));                   //数组a清0
   a[0]=s.length();                         //用a[0]计算字符串s的位数
   for(i=1;i<=a[0];i++)
          if(s[a[0]-i]>='0'&&s[a[0]-i]<='9')     a[i]=s[a[0]-i]-'0';
     else a[i]=s[a[0]-i]-'A'+10;
}
bool check(int a[])                         //判别整数数组a是否为回文数
{
   for(i=1;i<=a[0];i++)
          if(a[i]!=a[a[0]-i+1])return false;
   return true;
}
```

```
void  jia(int a[])                    //整数数组a与其反序数b进行n进制加法运算
{
    for(int i=1;i<=a[0];i++)b[i]=a[a[0]-i+1];      //反序数b
    for(int i=1;i<=a[0];i++) a[i]+=b[i];             //逐位相加
    for(int i=1;i<=a[0];i++)                        //处理进位
    {a[i+1]+=a[i]/n;
       a[i]%=n;
    }
    if(a[a[0]+1]>0) a[0]++;    //修正新的a的位数（a+b最多只能的一个进位）
}
int main()
{   init(a);
    if(check(a)){cout<<0<<endl;return 0;}
    ans=0;                                          //步数初始化为0
    while(ans++<=30)
    {       jia(a);
       if(check(a)){cout<<ans<<endl;return 0;}
    }
    cout<<"Impossible";                             //输出无解信息
    return 0;
}
```

【上机练习】

1、求N！的值

【问题描述】

用高精度方法，求N！的精确值(N以一般整数输入)。

【输入样例】ni.in

10

【输出样例】ni.out

3628800

2、求A/B高精度值

【问题描述】

计算A/B的精确值，设A，B是以一般整数输入，计算结果精确小数后20位**(若不足20位，末尾不用补0)** 。

【输入样例】ab.in

4 3

【输出样例】ab.out

4/3=1.33333333333333333333

【输入样例】ab.in

6 5

【输出样例】ab.out

6/5=1.2

3、求n累加和（ja）

【问题描述】

用高精度方法，求s=1+2+3+……+n的精确值(n以一般整数输入)。

【输入样例】ja.in

10

【输出样例】ja.out

55

4、阶乘和(sum)

【问题描述】

已知正整数N（N<=100）,设S=1!+2!+3!+...N!。其中"!"表示阶乘,即N!=1*2*3*……*(N-1)*N，如:3!=1*2*3=6。请编程实现：输入正整数N,输出计算结果S的值。

【输入样例】sum.in

4

【输出样例】sum.out

33

5、高精度求积(MULTIPLY)

【问题描述】

输入两个高精度正整数M和N（M和N均小于100位）。

【问题求解】

求这两个高精度数的积。

【输入样例】MULTIPLY.IN

36

3

【输出样例】MULTIPLY.OUT

108

6、天使的起誓（YUBIKILI.pas）

【问题描述】TENSHI非常幸运的被选为掌管智慧之匙的天使。在正式任职之前，她必须和其他新当选的天使一样，要宣誓。宣誓仪式是每位天使各自表述自己的使命，她们的发言稿被放在N个呈圆形排列的宝盒中。这些宝盒按顺时针方向被编上号码１、２、３……、N-1、N。一开始天使们站在编号为N的宝盒旁。她们各自手上都有一个数字，代表她们自己的发言稿所在的盒子是从１号盒子开始按顺时针方向的第几个。例如：有７个盒子，那么如果TENSHI手上的数字为９，那么她的发言稿所在盒子就是第２个。现在天使们开始按照自己手上的数字来找发言稿，先找到的就可以先发言。TENSHI一下子就找到了，于是她最先上台宣誓：“我将带领大家开启编程之门……”TENSHI宣誓结束以后，陆续有天使上台宣誓。可是有一位天使找了好久都找不到她的发言稿，原来她手上的数字M非常大，她转了好久都找不到她想找的宝盒。

【问题求解】

请帮助这位天使找到她想找的宝盒的编号。

【输入格式】

从文件YUBIKILI.IN的第一、二行分别读入正整数N和M，其中N、M满足

$2 \le N \le 10^8$，$2 \le M \le 10^{1000}$

【输出格式】

把所求宝盒的编号输出到文件YUBIKILI.OUT，文件只有一行（包括换行符）。

样例一

YUBIKILI.IN	YUBIKILI.OUT
7 9	2

样例二

YUBIKILI.IN	YUBIKILI.OUT
11 108	9

7、Hanoi双塔问题(Noip2007)

【问题描述】给定A、B、C三根足够长的细柱，在A柱上放有2*n*个中间有孔的圆盘，共有*n*个不同的尺寸，每个尺寸都有两个相同的圆盘，注意这两个圆盘是不加区分的（下图为*n*=3的情形）。现要将这些圆盘移到C柱上，在移动过程中可放在B柱上暂存。

要求：（1）每次只能移动一个圆盘；

（2）A、B、C三根细柱上的圆盘都要保持上小下大的顺序；

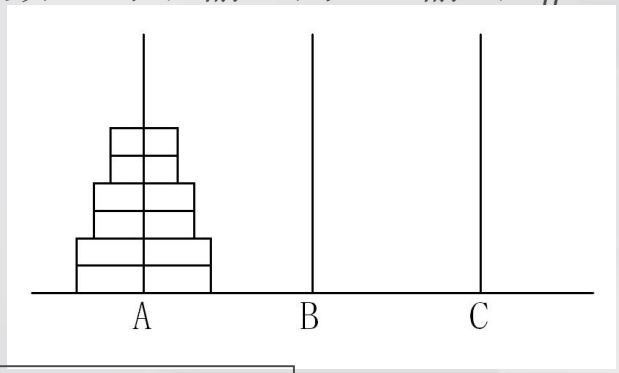
任务：设A_n为2*n*个圆盘完成上述任务所需的最少移动次数，对于输入的*n*，输出A_n。

【输入格式】

输入文件hanoi.in为一个正整数*n*，表示在A柱上放有2*n*个圆盘。

【输出格式】

输出文件hanoi.out仅一行，包含一个正整数, 为完成上述任务所需的最少移动次数A_n。



样例一

hanoi.in	hanoi.out
1	2

样例二

hanoi.in	hanoi.out
2	6

【限制】

对于50%的数据，1<=*n*<=25

对于100%的数据，1<=*n*<=200

【提示】设法建立A_n与A_{n-1}的递推关系式。