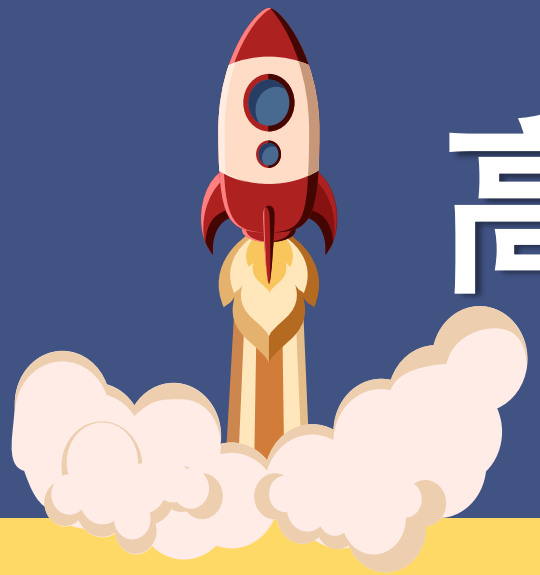


高精度计算

问题

【问题描述】

输入两个 1000 位以内的正整数，输出它们的和。

【问题说明】

在编程进行数值运算时，有时会遇到运算的精度要求特别高，远远超过各种数据类型的精度范围；有时数据又特别大，远远超过各种数据类型的极限值。

问题

【问题分析】

用字符串的方式读入两个高精度数，转存到两个整型数组 a 和 b 中，如图所示，模拟加法的过程，从低位(第 0 位)开始对应位 a[i] 和 b[i] 相加，同时处理进位，结果存储到另一个数组 c 中。最后，从高位到低位输出 c[i]。

	7	5	1	2	9	9	6	0	1	9
+					1	2	3	4	5	6
进位	0	0	0	1	1	0	0	0	1	0
	7	5	1	3	1	1	9	4	7	5

用字符串模拟竖式计算过程，即高精度计算。

高精度计算

高精度运算需要注意以下两点：

1. 要处理好数据的接收和存储问题；
2. 要处理好运算过程中的“进位”和“借位”问题。

高精度计算

1. 要处理好数据的接收和存储问题

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。

```
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
}
```

高精度计算

2. 要处理好运算过程中的“进位”和“借位”问题

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;

高精度加法

【例1】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a。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a

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

图b

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b所示。

高精度加法

程序设计如下：

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
char a1[100],b1[100];
int a[100],b[100],c[100],lena,lenb,lenc,i,x;
int main()
{
        scanf("%s",a1);
        scanf("%s",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]-'0';  //加数放入a数组
        for (i=0;i<=lenb-1;i++) b[lenb-i]=b1[i]-'0'; //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;
}
```

高精度减法

【例2】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 a[256],b[256],c[256],lena,lenb,lenc,i;
char n[256],n1[256],n2[256];
int main()
{
        scanf(“%s”,n1);
        scanf(“%s”,n2); //输入被减数与减数
```

高精度减法

```
if (strlen(n1)<strlen(n2)||(strlen(n1)==strlen(n2)&&strcmp(n1,n2)<0))
{//结果为负数的情况处理被减数和减数，交换被减数和减数
        strcpy(n,n1);
        strcpy(n1,n2);
        strcpy(n2,n);
        cout<<"-";
}
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;
}
```

高精度乘法

【例3】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a、图b。

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]+ 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a

$$\begin{array}{r} A_3\,A_2\,A_1 \\ \times\quad B_2\,B_1 \\ \hline C'_4\,C'_3\,C'_2\,C'_1 \\ C''_5\,C''_4\,C''_3\,C''_2 \\ \hline C_6\,C_5\,C_4\,C_3\,C_2\,C_1 \end{array}$$

图b

高精度乘法

高精度乘法的参考程序：

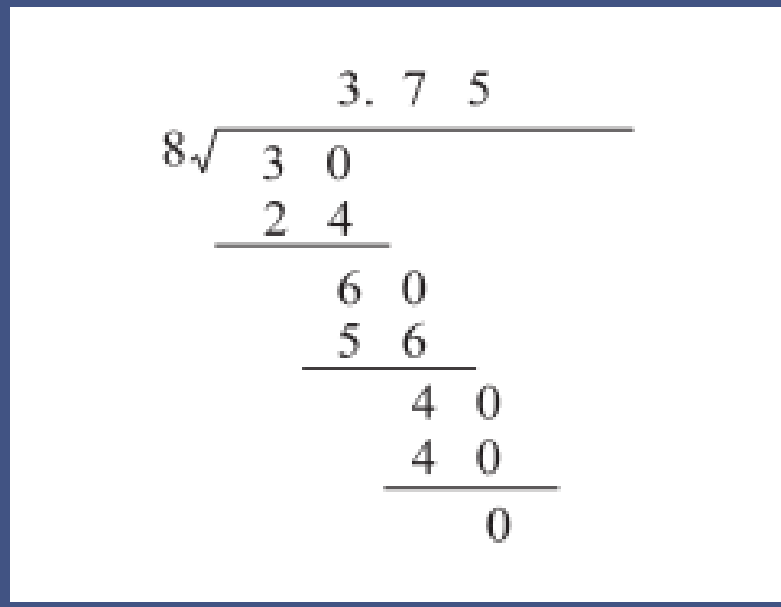
```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
char a1[100],b1[100];
int a[100],b[100],c[100],lena,lenb,lenc,i,j,x;
int main()
{
        scanf("%s",a1);
        scanf("%s",b1);
        lena=strlen(a1);
        lenb=strlen(b1);
        for (i=0;i<=lena-1;i++)
                a[lena-i]=a1[i]-'0';
         for (i=0;i<=lenb-1;i++)
                b[lenb-i]=b1[i]-'0';
for (i=1;i<=lena;i++)
{
    x=0; //用于存放进位
    for (j=1;j<=lenb;j++) //对乘数的每一位进行处理
    {
            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
             x=c[i+j-1]/10;
            c[i+j-1] %= 10;
    }
    c[i+lenb]=x;                        //进位
}
lenc=lena+lenb;
while (c[lenc]==0&&lenc>1)      //删除前导0
        lenc--;
for (i=lenc;i>=1;i--)
        cout<<c[i];
cout<<endl;
return 0;
```

高精度除法

【例4】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。



模拟模拟数学中的“短除法”。由数学知识可知，除法运算中被除数、除数和商、余数的关系为：

新的被除数 = 10× 余数

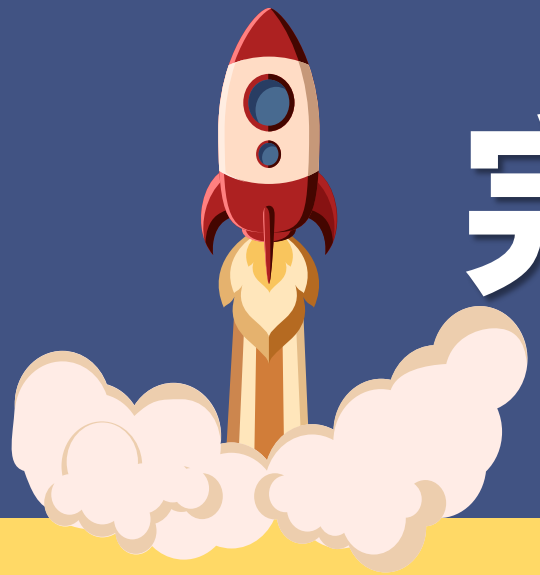
商 = 被除数 / 除数

余数 = 被除数 % 除数

高精度除法

高精度除以低精度参考程序：

```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
char a1[100],c1[100];
int a[100],c[100],lena,i,x=0,lenc,b;
int main()
{
        scanf(“%s”,a1);
        cin>>b;
        lena=strlen(a1);
        for (i=0;i<=lena-1;i++)
                a[i+1]=a1[i]-’0’;
        for (i=1;i<=lena;i++) //按位相除
        {
                c[i]=(x*10+a[i])/b;
                x=(x*10+a[i])%b;
        }
                lenc=1;
                while (c[lenc]==0&&lenc<lena)
                        lenc++; //删除前导0
                for (i=lenc;i<=lena;i++)
                cout<<c[i];
                cout<<endl;
                return 0;
}
```



完！

以梦为码 心之所往