算法是程序设计的核心，是灵魂，有了算法就可以结合数据结构转变成程序。秦九韶算法对应的程序如下：

```
#include <iostream>
using namespace std;
int n,a[20],x,ans;
int main()
{
    cin>>n;
    for(int i=0;i<=n;i++)cin>>a[i];
    cin>>x;
    ans=a[n];
    for(int i=1;i<=n;i++)ans=ans*x+a[n-i];
    cout<<ans<<endl;
    return 0;
}
```

7.2 高精度数值处理

高精度数值处理是采用模拟算法对位数达上百位甚至更多位数的数字进行各种运算，包括加法、减法、乘法、除法等基础运算。

在C++中，数值的加、减、乘、除运算都已经在系统内部被定义好了，我们可以很方便地对两个变量进行简单的运算。然而其中变量的取值范围，以整数为例，最大的是long long类型，范围是$[-2^{63},2^{63})$。假如我们要对两个范围更大，比如在$[-2^{1000},2^{1000})$范围内的数进行简单基础运算，就不能用C++的内部运算器了。同理，在使用实数类型计算时，由于精确度有限，也不能精确计算小数点后的数百位的数值。那么我们该如何对两个大整数进行运算呢？

回想一下我们小学数学课上的加法“竖式”运算。

首先把加数与被加数的个位对齐，然后个位对个位、十位对十位、百位对百位，位位对应进行加法操作，有进位的要相应地进行处理。

对应地，我们不妨对每一个数位开一个整数变量进行存储。那么两个位分别相加，进位这些问题我们都可以用程序表示出来。而在实践中，对进行运算的两个数分别用两个数组进行储存会使问题变得十分方便。

以上便是对“高精度加法”算法思路的简单描述。

如图7.2所示，我们要计算出537+543的值，不妨设数组A={7,3,5}，

B = {3,4,5}，我们还需要维护一个进位数组C，以及一个答案数组Ans。如同竖式，我们逐位计算：

（1）考虑第0位（个位），Ans[0] = A[0] + B[0] = 7 + 3 = 10，但我们需要保证Ans[0] < 10，那么在第0位就发生了进位，C[1] = 10 / 10 = 1，Ans[0]变为0。

（2）考虑第1位，Ans[1] = A[1] + B[1] = 3 + 4 = 7，但由于在第0位存在进位，那么Ans[1]还要加上C[1]，因此Ans[1] = 7 + C[1] = 8，8 < 10，因此在第1位没有发生进位。

（3）考虑第2位，Ans[2] = A[2] + B[2] + C[2] = 10，发生了进位，因此C[3] = Ans[2] / 10 = 1，Ans[2]变为0。

```
      5   3   7
  +   5   4   3
  ---------------
  1   0   8   0
    1       1
```

图7.2　加法竖式运算示意图

（4）考虑第3位，Ans[3] = A[3] + B[3] + C[3] = 1。

（5）程序结束，Ans = 1 * 1000 + 0 * 100 + 8 * 10 + 0 = 1080。

一般在进行高精度加法时，我们要考虑高位进位的情况，因此要注意数组范围。

对于高精度减法的算法与加法类似，但需要注意高位可能发生被减为0以及借位的情况，要重新计算答案的位数。如图7.3所示。

```
        1       1
      5   2   3
  -   4   3   7
  ---------------
      0   8   6
```

图7.3　减法竖式运算示意图

其实这两个算法的精华，我们在小学时就已接触过了，就是各种法则，比如“各数位对齐”、“逢10进1”、“小数减大数向高位借位”等。这些烂熟于心的法则，其实就是对这个算法的精简的表述。

【例7.1】高精度加法。现有两个10000位的正整数，要求你将它们加起来输出。

分析：算法流程基本上面已经给出了，要注意的是两个10000位的数

的和可能是10001位的数。

程序eg7.1如下：

```
1  //eg7.1
2  #include <iostream>
3  #include <string>
4  #include <algorithm>
5  using namespace std;
6  const int MAXN = 10005;
7  int A[MAXN],B[MAXN],C[MAXN],Ans[MAXN],Len_A,Len_B,Len_Ans;
8  void Read(int *A,int &Len)
9  {
10   string cur;
11   cin >> cur;
12   Len = cur.length();
13   for(int i = 0;i < Len;i ++) A[i] = cur[i] - 48;
14   reverse(A,A + Len);          // 对A[0]..A[Len-1]进行翻转
15 }
16 int main()
17 {
18   Read(A,Len_A);
19   Read(B,Len_B);
20   Len_Ans=max(Len_A,Len_B);
21   for(int i=0;i<=Len_Ans;i++)
22   {
23         Ans[i]=A[i]+B[i]+C[i];     // 加上前一位的进位值
24         if(Ans[i]>9)C[i+1]=Ans[i]/10,Ans[i]-= 10;
                                      // 处理进位
25   }
26   while(Ans[Len_Ans]>0)Len_Ans ++; // 位数是否增加
27   for(int i = Len_Ans - 1;i >= 0;i --)
28         cout << Ans[i];
29   return 0;
30 }
```

说明：在程序的第14行我们需要将读入的数组整体翻转（reverse函数是C++库中的翻转函数，传入的两个参数代表翻转的左闭右开区间），因为我们读入是从高位开始读入的。

【例7.2】高精度减法。现有两个10000位的正整数，要求你将它们做减法后输出，保证A>=B。

分析：减法操作与加法相似，但需要注意借位以及最终结果为0的情况。

程序eg7.2如下：

```
1  //eg7.2
2  #include <iostream>
3  #include <string>
4  #include <algorithm>
5  using namespace std;
6  const int MAXN = 10005;
7  int A[MAXN],B[MAXN],C[MAXN],Ans[MAXN],Len_A,Len_B,Len_Ans;
8  void Read(int *A,int &Len)
9  {
10   string cur;
11   cin >> cur;
12   Len = cur.length();
13   for(int i = 0;i < Len;i ++) A[i] = cur[i] - 48;
14   reverse(A,A + Len);
15 }
16 int main()
17 {
18   Read(A,Len_A);
19   Read(B,Len_B);
20   Len_Ans = max(Len_A,Len_B);
21   for(int i = 0;i < Len_Ans;i ++)
22   {
23       Ans[i] = A[i] - B[i] - C[i];
24       if(Ans[i]<0)C[i+1]++,Ans[i]+=10;// 借位操作
25   }
26   while(Len_Ans>1 && Ans[Len_Ans-1]==0)Len_Ans--;
                                        // 判断位数是否减少
27   for(int i = Len_Ans - 1;i >= 0;i --)
28        cout << Ans[i];
29   return 0;
30 }
```

【例7.3】高精度乘法。给定两个1000位的正整数A、B，求出A*B的值。

分析：高精度乘法比高精度加法要复杂一些，我们不妨先仿照竖式计

算模拟一遍。如图7.4所示。

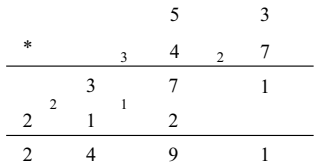

图7.4 乘法竖式运算示意图

观察图7.4，在计算53 * 47的值时，我们相当于将47拆为4 * 10 + 7来考虑，先计算出53*7，再计算出53*4，最后整体位移即可。一种更加数学化的语言是这样描述的：设A = (5 * 10 + 3)，B = (4 * 10 + 7)，那么A * B的值Ans根据乘法分配律有Ans = (5 * 4 * 100) + (5 * 7 + 3 * 4) * 10 + 3 * 7，那么我们可以每一位计算出其值，最后处理进位的情况即可。

比如说当前A = {5,3},B = {4,7},我们可以模拟这个过程，设有两个枚举变量i，j，表示当前A,B中扫描到哪个位置。

（1）i = 0，j = 0，Ans[i+j] = Ans[0] = A[0]*B[0] = 21。

（2）i = 0，j = 1，Ans[i+j] = Ans[1] = A[0]*B[1] = 12。

（3）i = 1，j = 0，Ans[i+j] = Ans[1] = Ans[1] + A[1] * B[0] = 12 + 35 = 47。

（4）i = 1，j = 1，Ans[i+j] = Ans[2] = A[1] * B[1] = 20。

（5）Ans = {20,47,21}，从低位向高位处理进位，最终得到Ans = {2,4,9,1}。

这个算法的精髓就是将一个数看成若干个10的次幂的和，利用乘法分配律各个计算，最终再处理进位的问题，更具体的细节可以看下面的代码。

```
1  //eg7.3
2  #include <iostream>
3  #include <string>
4  #include <algorithm>
5  using namespace std;
6  const int MAXN = 5005;
7  int A[MAXN],B[MAXN],Ans[MAXN],Len_A,Len_B,Len_Ans;
8  void Read(int *A,int &Len)
9  {
10   string cur;
11   cin >> cur;
12   Len = cur.length();
```

```
13   for(int i = 0;i < Len;i ++) A[i] = cur[i] - 48;
14   reverse(A,A + Len);
15 }
16 int main()
17 {
18   Read(A,Len_A);
19   Read(B,Len_B);
20   Len_Ans = Len_A + Len_B - 1;
21   for(int i = 0;i < Len_A;i ++)
22     for(int j = 0;j < Len_B;j ++)
23        Ans[i + j] += A[i] * B[j];
24   for(int i = 0;i < Len_Ans;i ++)
25     if(Ans[i]>9)Ans[i+1]+=Ans[i]/10,Ans[i]%=10;
                                          // 最后进行进位处理
26   while(Ans[Len_Ans])Len_Ans ++;
27   for(int i = Len_Ans - 1;i >= 0;i --)
28     cout << Ans[i];
29   return 0;
30 }
```

注意：两个长度为n的正整数的乘积的位数可能达到2*n。

【例7.4】高精度除法。给定两个1000位的正整数A、B，求出A/B的商和余数。

分析：高精度除法同样采用竖式除法的方法，竖式除法如图7.5所示。

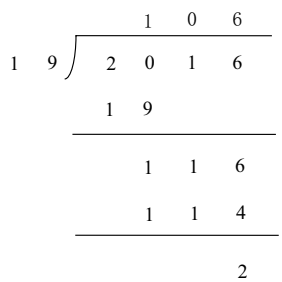

图7.5　竖式除法运算示意图

高精度除法从被除数的高位到低位依次计算出商对应位上的数字，记录在除法过程中的余数为C，高精度除法就转变为C除以B的操作，除法操作转换为减法操作，用C减去B，每减一次对应位上的商加1，直到不够减为止，再处理下一位，先产生新的余数C=C*10+A[i]，再重复上述操作。

以样例为例介绍高精度除法的过程：

（1）i=3，余数C=2，B=19，商对应的位的值Ans[3]=0。

（2）i=2，C=C*10+A[2]=20，可以从C减去B一次，因此Ans[2]=1，C=1。

（3）i=1，C=C*10+Ans[1]=11，不够减B，Ans[1]=0。

（4）i=0，C=C*10+Ans[0]=116，可以从C中减去B共6次，因此Ans[0]=6，C=2。

（5）至此，除法结束。Ans=106，C=2。

对应的程序eg7.4如下：

```
1  //eg7.4
2  #include <iostream>
3  #include <string>
4  #include <algorithm>
5  using namespace std;
6  const int MAXN = 5005;
7  int A[MAXN],B[MAXN],Ans[MAXN],C[MAXN],Len_A,Len_B,Len_Ans;
8  void Read(int *A)
9  {
10   string cur;
11   cin >> cur;
12   A[0] = cur.length();
13   for(int i = 1;i <=A[0];i ++) A[i] = cur[i-1] - 48;
14   reverse(A+1,A + A[0]+1);
15 }
16 bool Big()           // 比较余数C与除数B的大小，如果C>=B，
                          则返回true，否则返回false
17 {
18   if(C[0]>B[0]) return true;
19   if(C[0]<B[0]) return false;
20   for(int i=C[0];i>=1;i--)
21     if(C[i]<B[i]) return false;
22     else if (C[i]>B[i]) return true;
23   return true;
24 }
25 void Minus()         // 从C中减去B
26 {
27   int c=0;
28   for(int i=1;i<=C[0];i++)
```

```
29   {
30     C[i]-=B[i]+c;
31     if(C[i]<0){C[i]+=10;c=1;}
32     else c=0;
33   }
34   while(C[0]>1 && C[C[0]]==0) C[0]--;
35 }
36 void output(int *A)
37 {
38   for(int j=A[0];j>=1;j--)cout<<A[j];
39   cout<<endl;
40 }
41 int main()
42 {
43   Read(A);
44   Read(B);
45   C[0]=0;
46   for(int i=A[0];i>=1;i--)
47   {
48     for(int j=C[0];j>=1;j--)C[j+1]=C[j];
49     C[1]=A[i];
50     C[0]++;
51     while(Big())
52     {
53       Minus();
54       Ans[i]++;
55     }
56   }
57   Ans[0]=A[0];
58   while(Ans[0]>1 && Ans[Ans[0]]==0) Ans[0]--;
59   output(Ans);
60   output(C);
61   cin>>A[0];
62   return 0;
63 }
```

7.3　简单枚举算法

枚举，也叫做穷举法。从字面意思来理解，就可以知道是指列出所有