

# 无限长整数计算器上机报告

数院：吕世杰 学号：00401060

日期：2005 11 16

摘要：一 问题的描述和分析

五 主要算法及时间分析

二 程序的具体使用方式

六 评价和进一步工作

三 算法的设计

四 数据结构的设计

## 一 问题的描述和分析

编制一个能计算无限长整数之间的加法，减法，乘法，除法（整除），以及乘方的计算。能实现一个计算器的简化功能：

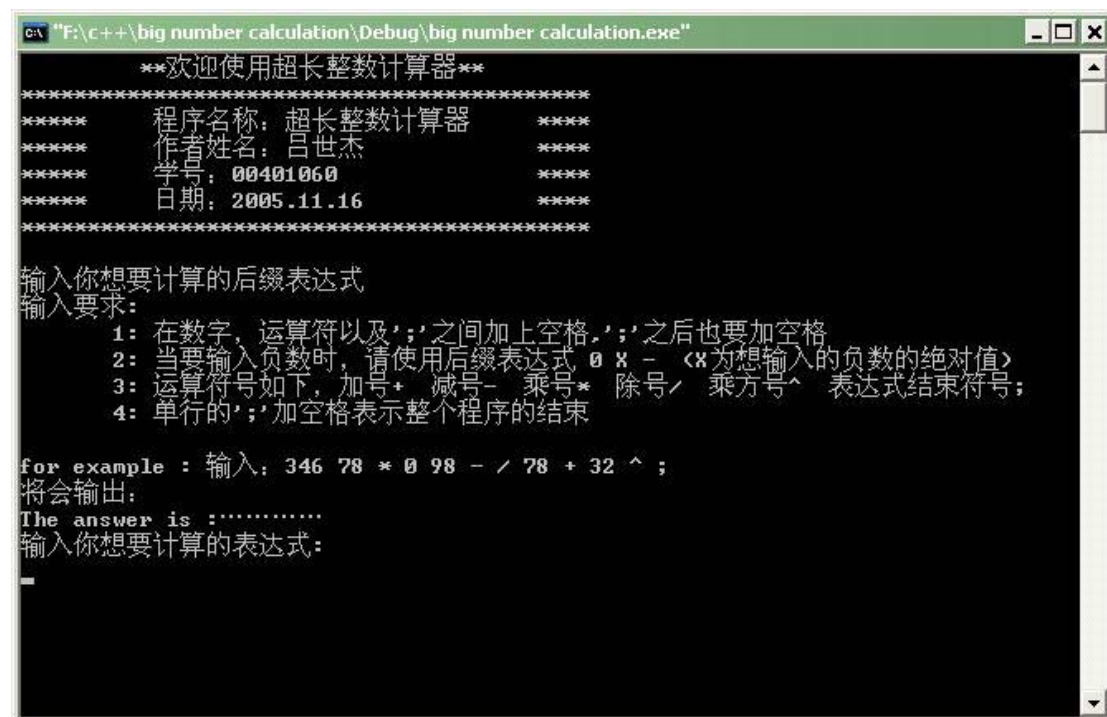
比方说可以计算数位很长很长（理论上可以无限长）的两个整数之间的普通复合运算。输入的表达式方式为后缀表达式，包含正整数以及负整数之间的基本运算。其中负整数要求操作者以（0 X --）的后缀减法形式输入，这是因为表达式本来就没特别固定的要求，并且所有负数均可看作减法运算，因此并没有为负数形式的读入作特别的程序。还有就是除法的设计由于很仓促，因此若被除数位数与除数位数相比过大时会算得很慢。希望操作者不会对此介意。以上缺点都将会在报告中提出如何改进。

操作界面为键盘输入方式。一开始不需要空格的输入你想要计算的后缀表达式，在连续的输入数字后，在与下一个数或符号之间都要加上空格，并且符号之间也要加空格，表达式以‘;’作为结束，但也要在其之后加上空格，每输入一个表达式都会马上计算出结果并打印出来。单行的分号加空格表示退出。

## 二 程序的具体使用方式

输入格式已经在上一段中具体介绍过了，但要注意程序中用的整数是 int 型的，因此位数在输入和输出时都有限制，如果遇到位数超过 32767 位时，可将程序中的 int 换作 long int，此时位数的表达又将上一个新台阶。

具体例子：按下执行程序后弹出的界面为下图



```
C:\ "F:\c++\big number calculation\Debug\big number calculation.exe"

**欢迎使用超长整数计算器**
*****
**** 程序名称: 超长整数计算器 ****
**** 作者姓名: 吕世杰 ****
**** 学号: 00401060 ****
**** 日期: 2005.11.16 ****
*****

输入你想要计算的后缀表达式
输入要求:
1: 在数字, 运算符以及';'之间加上空格,';'之后也要加空格
2: 当要输入负数时, 请使用后缀表达式 0 x - <x为想输入的负数的绝对值>
3: 运算符号如下, 加号+ 减号- 乘号* 除号/ 乘方号^ 表达式结束符号;
4: 单行的';'加空格表示整个程序的结束

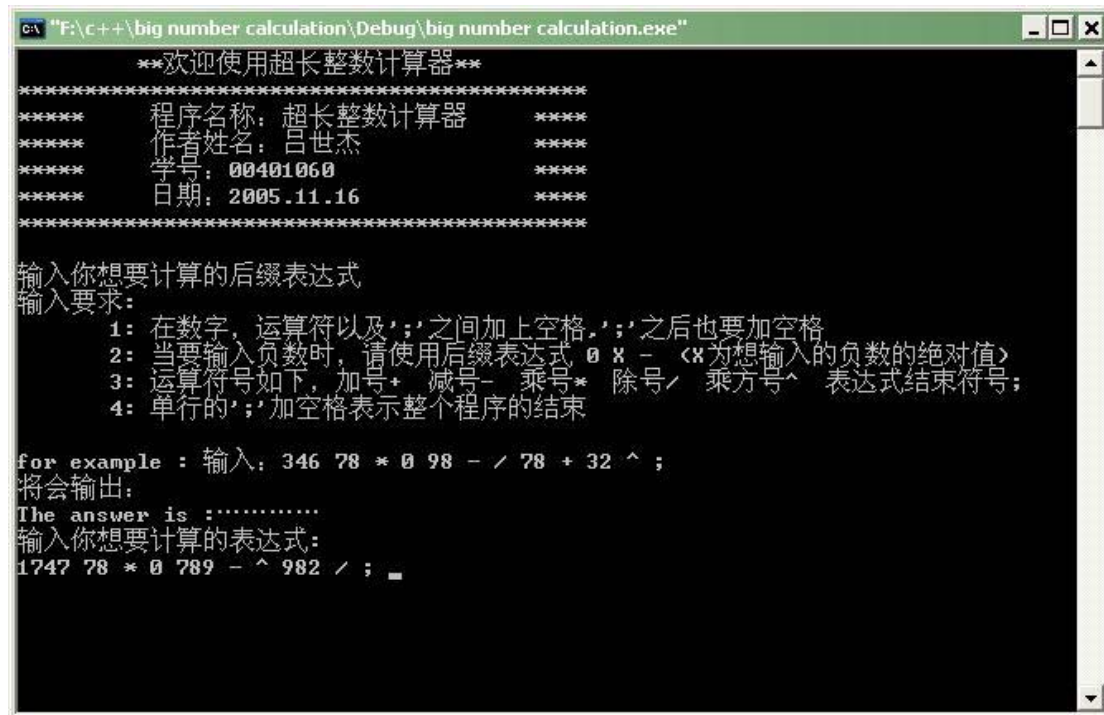
for example : 输入: 346 78 * 0 98 - / 78 + 32 ^ ;
将会输出:
The answer is :.....
输入你想要计算的表达式:
-
```

操作者可以输入表达式：

比如 1747@78@\*@0@789@-@^@982@/@;@

(@表示空格)

相当于： $(1747 * 78)^{-789} / 982$  应该等于 0。（因为设定了一个数若不是 0，1，-1，那么它的负数次方均等于 0）



```
C:\ "F:\c++\big number calculation\Debug\big number calculation.exe"

**欢迎使用超长整数计算器**
*****
***** 程序名称：超长整数计算器 *****
***** 作者姓名：吕世杰 *****
***** 学号：00401060 *****
***** 日期：2005.11.16 *****
*****

输入你想要计算的后缀表达式
输入要求：
1: 在数字，运算符以及';'之间加上空格，';'之后也要加空格
2: 当要输入负数时，请使用后缀表达式 0 x - <x为想输入的负数的绝对值>
3: 运算符号如下，加号+ 减号- 乘号* 除号/ 乘方号^ 表达式结束符号;
4: 单行的';'加空格表示整个程序的结束

for example : 输入: 346 78 * 0 98 - / 78 + 32 ^ ;
将会输出:
The answer is :.....
输入你想要计算的表达式:
1747 78 * 0 789 - ^ 982 / ; _
```

此时屏幕会打印出：



```
C:\ "F:\c++\big number calculation\Debug\big number calculation.exe"

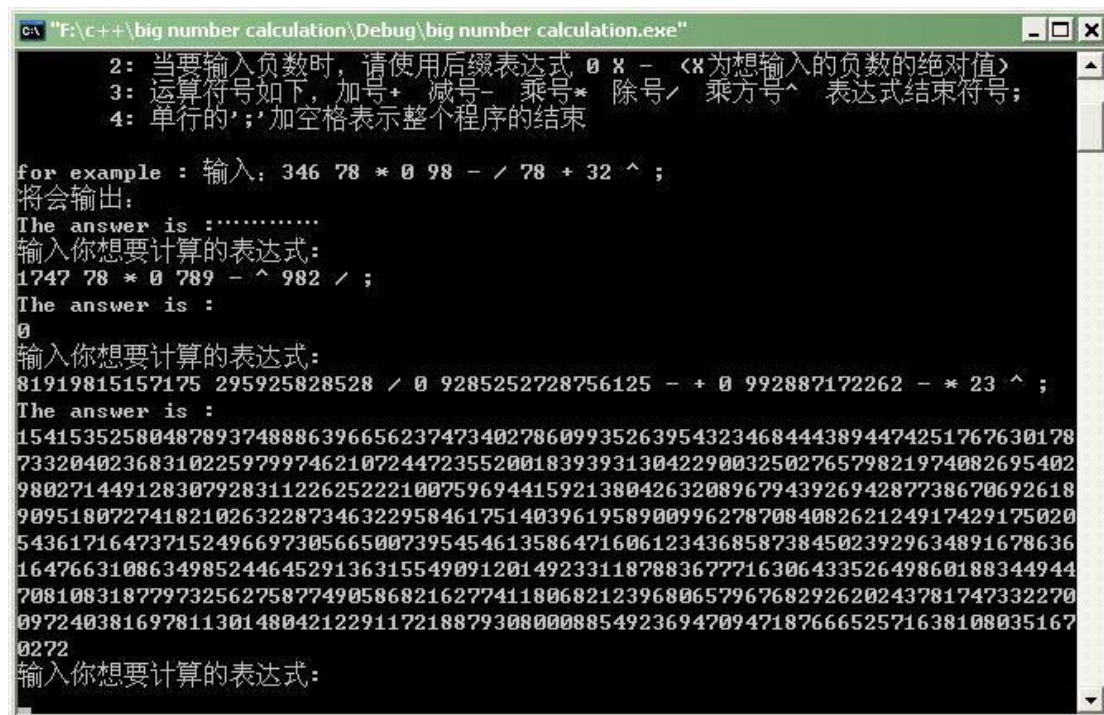
**欢迎使用超长整数计算器**
*****
***** 程序名称：超长整数计算器 *****
***** 作者姓名：吕世杰 *****
***** 学号：00401060 *****
***** 日期：2005.11.16 *****
*****

输入你想要计算的后缀表达式
输入要求：
1: 在数字，运算符以及';'之间加上空格，';'之后也要加空格
2: 当要输入负数时，请使用后缀表达式 0 x - <x为想输入的负数的绝对值>
3: 运算符号如下，加号+ 减号- 乘号* 除号/ 乘方号^ 表达式结束符号;
4: 单行的';'加空格表示整个程序的结束

for example : 输入: 346 78 * 0 98 - / 78 + 32 ^ ;
将会输出:
The answer is :.....
输入你想要计算的表达式:
1747 78 * 0 789 - ^ 982 / ;
The answer is :
0
输入你想要计算的表达式:
_
```

继续测试数据

输入: 81919815157175@295925828528@/@0@9285252728756125@-@+@0 @992887172262  
@-@\*@23@^@;@  
相当于 $[(8191815157175 / 295925828528) + (-9285252728756125) * (-992887172262)]^{23}$   
 $= 1.5415352580497401699483699303488e+643$  (用计算器计算的结果)



```
2: 当要输入负数时, 请使用后缀表达式 0 x - <x为想输入的负数的绝对值>
3: 运算符如下, 加号+ 减号- 乘号* 除号/ 乘方号^ 表达式结束符号;
4: 单行的'; 加空格表示整个程序的结束

for example : 输入: 346 78 * 0 98 - / 78 + 32 ^ ;
将会输出:
The answer is : .....
输入你想要计算的表达式:
1747 78 * 0 789 - ^ 982 / ;
The answer is :
0
输入你想要计算的表达式:
81919815157175 295925828528 / 0 9285252728756125 - + 0 992887172262 - * 23 ^ ;
The answer is :
15415352580487893748886396656237473402786099352639543234684443894474251767630178
73320402368310225979974621072447235520018393931304229003250276579821974082695402
98027144912830792831122625222100759694415921380426320896794392694287738670692618
90951807274182102632287346322958461751403961958900996278708408262124917429175020
54361716473715249669730566500739545461358647160612343685873845023929634891678636
16476631086349852446452913631554909120149233118788367771630643352649860188344944
70810831877973256275877490586821627741180682123968065796768292620243781747332270
09724038169781130148042122911721887930800088549236947094718766652571638108035167
0272
输入你想要计算的表达式:
```

继续测试数据

输入: 0@89@-@9@^@0@1987@-@/@2234@+@13\* @;@  
相当于 $((-89)^9 / (-1987) + 2234) * 13 = 2292216028311521$  (用计算器计算的结果)



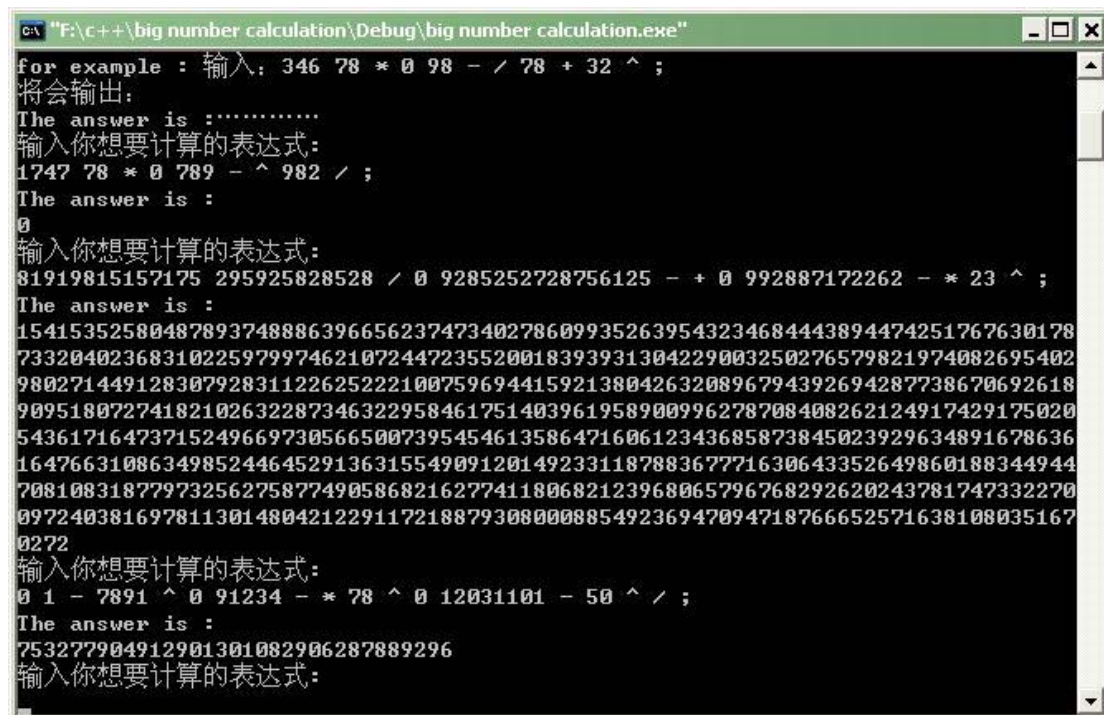
```
for example : 输入: 346 78 * 0 98 - / 78 + 32 ^ ;
将会输出:
The answer is : .....
输入你想要计算的表达式:
1747 78 * 0 789 - ^ 982 / ;
The answer is :
0
输入你想要计算的表达式:
81919815157175 295925828528 / 0 9285252728756125 - + 0 992887172262 - * 23 ^ ;
The answer is :
15415352580487893748886396656237473402786099352639543234684443894474251767630178
73320402368310225979974621072447235520018393931304229003250276579821974082695402
98027144912830792831122625222100759694415921380426320896794392694287738670692618
90951807274182102632287346322958461751403961958900996278708408262124917429175020
54361716473715249669730566500739545461358647160612343685873845023929634891678636
16476631086349852446452913631554909120149233118788367771630643352649860188344944
70810831877973256275877490586821627741180682123968065796768292620243781747332270
09724038169781130148042122911721887930800088549236947094718766652571638108035167
0272
输入你想要计算的表达式:
0 89 - 9 ^ 0 1987 - / 2234 + 13 * ;
The answer is :
2292216028311521
输入你想要计算的表达式:
```



继续测试数据

输入: 0@1@-@7891@^0@91234@-@\*78@^@0@12031101@-@50@^@/@;@

相当于表达式:  $(-1)^{7891} * (-91234)^{78} / ((-12031101)^{50}) = 7.532779049129013010829062878893e+32$  (用计算器计算的结果)

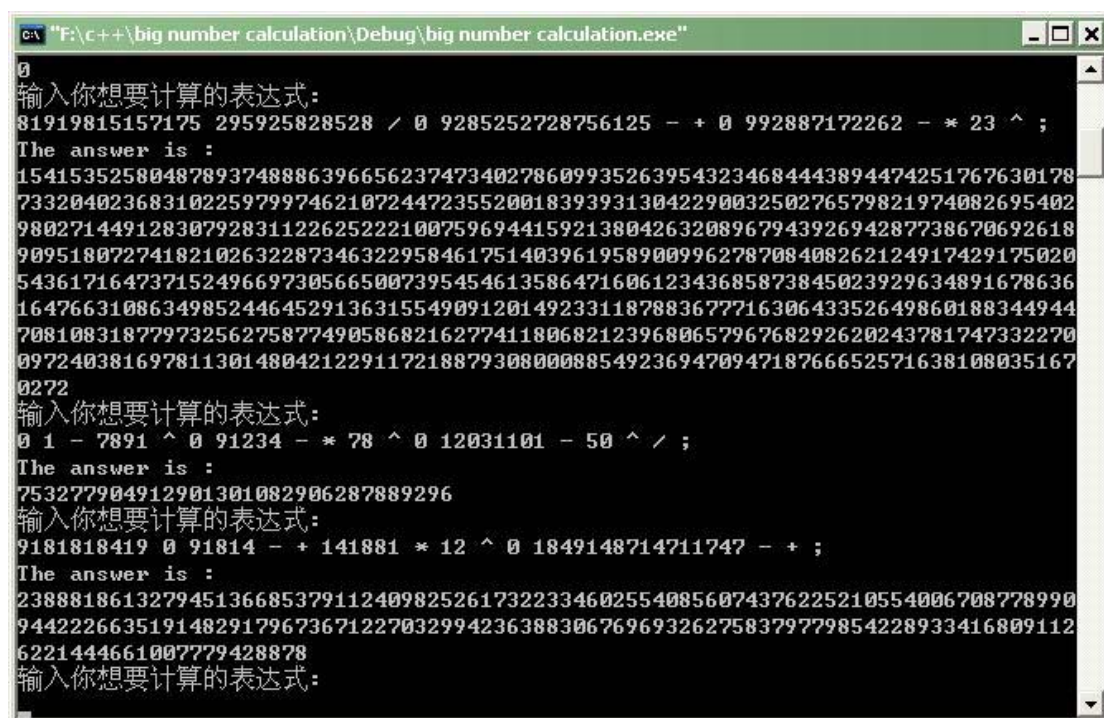


```
C:\ "F:\c++\big number calculation\Debug\big number calculation.exe"
for example : 输入: 346 78 * 0 98 - / 78 + 32 ^ ;
将会输出:
The answer is :.....
输入你想要计算的表达式:
1747 78 * 0 789 - ^ 982 / ;
The answer is :
0
输入你想要计算的表达式:
81919815157175 295925828528 / 0 9285252728756125 - + 0 992887172262 - * 23 ^ ;
The answer is :
15415352580487893748886396656237473402786099352639543234684443894474251767630178
73320402368310225979974621072447235520018393931304229003250276579821974082695402
98027144912830792831122625222100759694415921380426320896794392694287738670692618
90951807274182102632287346322958461751403961958900996278708408262124917429175020
54361716473715249669730566500739545461358647160612343685873845023929634891678636
16476631086349852446452913631554909120149233118788367771630643352649860188344944
70810831877973256275877490586821627741180682123968065796768292620243781747332270
09724038169781130148042122911721887930800088549236947094718766652571638108035167
0272
输入你想要计算的表达式:
0 1 - 7891 ^ 0 91234 - * 78 ^ 0 12031101 - 50 ^ / ;
The answer is :
753277904912901301082906287889296
输入你想要计算的表达式:
```

继续测试数据

输入: 9181818419@0@91814@-@+@141881@\*12@^@0@1849148714711747@-@+@;@

相当于表达式:  $[(9181818419 + (-91814)) * 141881]^{12} + (-1849148714711747) = 2.3888186132794513668537911240983e+181$  (用计算器计算的结果)



```
C:\ "F:\c++\big number calculation\Debug\big number calculation.exe"
0
输入你想要计算的表达式:
81919815157175 295925828528 / 0 9285252728756125 - + 0 992887172262 - * 23 ^ ;
The answer is :
15415352580487893748886396656237473402786099352639543234684443894474251767630178
73320402368310225979974621072447235520018393931304229003250276579821974082695402
98027144912830792831122625222100759694415921380426320896794392694287738670692618
90951807274182102632287346322958461751403961958900996278708408262124917429175020
54361716473715249669730566500739545461358647160612343685873845023929634891678636
16476631086349852446452913631554909120149233118788367771630643352649860188344944
70810831877973256275877490586821627741180682123968065796768292620243781747332270
09724038169781130148042122911721887930800088549236947094718766652571638108035167
0272
输入你想要计算的表达式:
0 1 - 7891 ^ 0 91234 - * 78 ^ 0 12031101 - 50 ^ / ;
The answer is :
753277904912901301082906287889296
输入你想要计算的表达式:
9181818419 0 91814 - + 141881 * 12 ^ 0 1849148714711747 - + ;
The answer is :
23888186132794513668537911240982526173223346025540856074376225210554006708778990
94422266351914829179673671227032994236388306769693262758379779854228933416809112
6221444661007779428878
输入你想要计算的表达式:
```

若需要退出就输入: ;@

```

c:\f:\c++\big number calculation\Debug\big number calculation.exe
The answer is :
15415352580487893748886396656237473402786099352639543234684443894474251767630178
73320402368310225979974621072447235520018393931304229003250276579821974082695402
98027144912830792831122625222100759694415921380426320896794392694287738670692618
90951807274182102632287346322958461751403961958900996278708408262124917429175020
54361716473715249669730566500739545461358647160612343685873845023929634891678636
16476631086349852446452913631554909120149233118788367771630643352649860188344944
70810831877973256275877490586821627741180682123968065796768292620243781747332270
09724038169781130148042122911721887930800088549236947094718766652571638108035167
0272
输入你想要计算的表达式:
0 1 - 7891 ^ 0 91234 - * 78 ^ 0 12031101 - 50 ^ / ;
The answer is :
753277904912901301082906287889296
输入你想要计算的表达式:
9181818419 0 91814 - + 141881 * 12 ^ 0 1849148714711747 - + ;
The answer is :
23888186132794513668537911240982526173223346025540856074376225210554006708778990
94422266351914829179673671227032994236388306769693262758379779854228933416809112
6221444661007779428878
输入你想要计算的表达式:
;
The calculation is over.
谢谢使用本系统!
Press any key to continue

```

### 三 算法的设计

算法设计就是编写加法, 乘法, 减法, 乘方, 和除法。具体的每个算法将于算法的分析里面进行陈述。当然还包括编写读入和输出函数。读入函数一读到空格为标志, 因此我的程序在每个输入的东西之后都要加空格。我使用的加法都很普通, 数字也是一位数字一位数字地进行读取。其实可以将读入的数转为二进制形式, 动态顺序表的每一位里存 16 位数字。这样将大大加速程序, 无论在时间和空间上都优化了。

### 四 数据结构的设计



表达式采取了后缀的形式, 因此使用 PLinkStack 类型的栈来存放元素, 超长整数运用的是动态顺序表来实现存储, 栈中存放的也正是动态顺序表。以动态顺序表的最后一位是否是数字来判断读入函数得到的表是否需要压入栈中, 若读到的是运算符算代表的数字, 那么从栈中取出两个元素, 进行相对应的操作, 并把结果存入栈中, 这个过程是通过一个叫 calculator 的程序来实现的。之后的 getandput 整个输入----运算-----输出的函数结构, 并且里面附带了对于: 结束时是否打印答案以及是否退出的判断。它的返回值时 0 或 1, 方便在主程序 operation 中的循环语句进行判断。另外就是动态顺序表每一位只存入一个整形数字, 这一点没有充分利用空间, 还值得改进。

### 五 主要算法及时间分析

#### 1. 加法和减法

加法和减法两个函数都一起放在了 add and minus.cpp 中了

现在此声明一点, 我所有的算法都考虑的绝对值的加减乘除以及乘方, 我在动态顺序表 PSeqList 这个结构指针的内部写了一个 PSeqList->c, 用来存放加号或负号, 初始都定义为加号。在做具体运算时只需判断一下两个变量的符号来决定最后返回值的符号即可。输出函数中有符号的判断。

加法函数形式是 PSeqList add(PSeqList lp1,PSeqList lp2 )

现设 lp1 是 n 位数，lp2 是 m 位数。主要程序如下：

```
int x,y,z,i=0,ad=0;
while ((lp1->n)>(lp2->n))
    insert_seq(lp2,lp2->n,0);
while ((lp2->n)>(lp1->n))
    insert_seq(lp1,lp1->n,0);
for(;i<(lp1->n);i++)
{
    x=lp1->elems[i];
    y=lp2->elems[i];
    z=(x+y+ad)%10;
    ad=((x+y+ad)-z)/10;
    lp1->elems[i]=z;
}
if(ad>0)
    insert_seq(lp1,lp1->n,ad);
free_1(lp2);
return lp1;
```

首先判断哪个数比较大（当然是绝对值大小），然后把短的数前面插 0 补齐，此步骤操作为 n-m（不妨认为 n>m），然后下面的 for 语句进行了 n 位操作，这个 for 中进行的是每一位数相应的加法，ad 表示的是进位得到的数字，为 1 或 0。左后一个判断语句是决定最后进不进位。最终 add 将改变 lp1，并返回 lp1 作为加法得到的结果。所以总操作为  $T(n)=O(n)$ ，即时间复杂性为  $O(n)$ 。

减法函数形式是 PSeqList minus(PSeqList lp1,PSeqList lp2 )

现设 lp1 是 n 位数，lp2 是 m 位数。主要程序如下：

```
lp2->elems[0]=10-lp2->elems[0];
for(i=1;i<lp2->n;i++)
    lp2->elems[i]=9-lp2->elems[i];
j=lp2->n;
add(lp1,lp2);
for(i=j;i<lp1->n;i++)
{
    if(lp1->elems[i]>0)
    {
        lp1->elems[i]=lp1->elems[i]-1;
        while(lp1->elems[lp1->n-1]==0&&lp1->n>1)/*就是说减下来的数
形式是 09999.....,那么把首位去掉*/
        {
            delete_seq(lp1,lp1->n-1);
        }
    }
}
return(lp1);
```

```

    }
    else
        lp1->elems[i]=9;
}

```

想法如下：主要是要调用加法运算来帮助实现。比如  $198374-9823$ ，我计算的其实是  $198374-(10000-177)=193874+177-10000$

一开始对  $lp2$  的改变其实就是上面写的如何得到 177，操作了  $m$  步，然后进行加法，再考虑进位问题以及得到的数前几位都是 0 的情况。总之，减法跟加法没多大区别，也是将第一个变量  $lp1$  改编成结果返回。时间复杂度  $T(n)=O(n)$ 。

## 2. 乘法与乘方

乘法与乘方被放在了 `multiple and power.cpp` 文件中

乘法函数形式 `PSeqList multiple(PSeqList lp1, PSeqList lp2)`

里面有个 `multiple_unit(PSeqList lp1, int x)`，是将顺序表与一位数的乘法。

乘法算法大致如下：

首先设被乘数  $lp1$  为  $n$  位数， $lp2$  为  $m$  位数。

通过 `multiple_unit` 将  $lp1$  与 0-9 的乘积存放在一个动态顺序表数组 `s[10]` 中，以后就不用乘了，只需根据  $lp2$  的每一位进行选取即可。不妨假设计算的是  $18927566*1988$ ，那么先将  $18927566$  与 0-9 的乘积存在一个数组里，这进行了  $10n$  次操作，然后让一个动态表指针  $lp$  指向  $18927566*8=151420528$ ，然后根据 1988 十位数是 8 选取 15142058 与原来值进行错位相加：

```

      1 5 1 4 2 0 5 2 8
+ 1 5 1 4 2 0 5 2 8

```

先将下面那个数的第一位 1 复制到上面那个数中，此时变成

```

      1 1 5 1 4 2 0 5 2 8
+ 1 5 1 4 2 0 5 2 8

```

然后从下面那个数的个位到 5 那位与上面的数进行加法，一共进行了  $n$  次，然后再考虑进位情况。

接下来将得到的数 1665625808 与 18927566 与 9 的乘积 170348094 进行错位相加

```

      1 6 6 5 6 2 5 8 0 8
+ 1 7 0 3 4 8 0 9 4

```

同样进行  $n$  次计算。

接下去继续进行错位相加。所以一共需要的步骤大致为  $10n+n*m$  次，时间复杂度  $T(n,m)=O(n*m)$

## 2 乘方式通过三个函数来实现的，分别是 `PSeqList power(PSeqList lp1, PSeqList lp2)`

`PSeqList power_1(PSeqList lp1, int n)` `PSeqList power_2(PSeqList lp1, int n)`

其中第一个是实现零个动态顺序表的乘方计算，即  $lp1$  的  $lp2$  次方。`power_1` 实现的是当  $n$  为 0-10 时的乘方，`power_2` 实现的是计算  $lp1$  的 10 的  $n$  次方数。想法就是让  $lp1$  的十次方作为一个基本的量，然后就可以通过 `power_2` 来实现  $lp1$  的 10 的  $n$  次方。如果只是让  $lp1$  累乘，速度太慢，因此可以通过  $lp2$  的每一位数，来实现  $lp1$  的 10 的  $n$  次方累乘相应遍，这样会比较快一些。比方说为了得到 2 的 6897 次方，可以先用 `power_1` 计算 2 的 7 次方，存入结果，然后做 9 次 2 的 10 次方，与原来结果相乘存入结果，再做 8 次 2 的 100 次方，再相乘存入结果，最后作 6 次 2 的 1000 次方，并得到最后的结果。



大致算法如下：

```
LP1=power_1(lp1,lp2->elems[0]);
for(i=1;i<lp2->n;i++)
{
    for(j=0;j<lp2->elems[i];j++)
    {
        lp=multiple(LP1,power_2(lp1,i));
        free_1(LP1);
        LP1=lp;
    }
}
```

现在开始设  $lp1$  为  $n$  位数,  $lp2$  为  $m$  位数。现考虑  $power\_1$  的操作步骤, 最多算 10 次累乘, 操作数为  $n^2+2n^2+3n^2+\cdots+9n^2=45n^2$  次。再考虑  $power\_2(lp1,a)$ , 大致需要  $45n^2*a$  次步骤。最后考虑  $power$  的时间代价。对于  $lp2$  的第  $a$  位, 最多算 10 次, 故总步骤大致等于  $45n^2*10*(a+a-1+\cdots+3+2+1)*10^{2a} \sim C*n^2*a^2*10^{2a}$ , 其中  $C$  为一个常数。

又  $lp2$  的位数大小约等于  $\lg m$ , 因此, 总时间代价约为  $(n*m*\lg m)^2$ 。即时间代价  $T(n,m)=O((n*m*\lg m)^2)$ 。

利用所写程序 `testime.cpp`,

可以对 2 的 10000 次方进行测速, 得到结果 290。



再测试 2 的 20000 次方进行测速, 得到结果 1151。

再测试 2 的 30000 次方进行测速, 得到结果 2593。

再测试 2 的 40000 次方进行测速, 得到结果 4637。

再测试 2 的 50000 次方进行测速, 得到结果 7181。

若测试 2 的 100000 次方进行测速, 得到结果 26498。

对 4 的 10000 次方进行测速, 得到结果 1161。

对 4 的 20000 次方进行测速, 得到结果 4696。

对 4 的 30000 次方进行测速, 得到结果 10826。

对 4 的 40000 次方进行测速, 得到结果 19108。

对 4 的 50000 次方进行测速, 得到结果 29863。

由上述数据可见, 基本上时间分析没有错。

### 3 除法

除法设计的比较差, 形式为 `PSeqList divide(PSeqList lp1, PSeqList lp2)` 在 `divide.cpp` 文件中。主要思想如下, 将除数不断扩大两倍去与被除数比较, 当除数乘以 2 的  $n$  次幂比被除数小并且  $n+1$  次幂比被除数大时, 将 2 的  $n$  次幂记入商, 然后将被除数减去除数乘以 2 的  $n$  次幂 继续与除数比较, 比较过程与上面相同, 都是不断扩大两倍的比较。

## 六 评价和进一步工作



由于使用了动态顺序表, 因此在空间上消耗比较大, 且容易进行计算。如今虽已经做完, 发现每一位只存入一个整数非常亏。如果能以二进制表示来读入数字, 并在数组的每一位存一个 15 位长度的二进制的数, 将会好很多。另一方面, 如果每次读数每位都读 4 位数字, 将一个 4 位数放入数组中的一个里面, 也会快很多, 尤其做乘法, 可以直接使用程序自带的乘法运算, 将方便很多。至于除法设计, 可以将被除数取前几位, 除数也取前几位, 然后用程



序自带的除法做商进行估计，这样将会快很多很多，除法的设计中由于仓促，以此计算位数相差很多的运算将很慢，如果做商估计将便利很多。

总之，通过这次大程序的编写，学会了很多，遇到了很多困难，通过不停的调试基本符合了作业要求，收获很大，尤其是一开始写乘方，算法很慢，因为用的是累乘，后来在乘法以及乘方本身都作了些改进，比原先快了很多了，虽然不足之处仍很多，但这次程序设计的经验和收益很大，以后将想得更周到些。

数院：吕世杰

学号：00401060

日期：2005 11 16