**实验题目： 大整数的运算**

**班级： 软件工程05班**

**姓名： 杨归一**

**学号： 201806062327**

**完成日期： 2019年11月23日**

**目录**

1. **实验题目和要求……………………………………… 3**

**二、 设计思路……………………………………………… 4**

1. 系统总体设计………………………………………… 4

2. 系统功能设计………………………………………… 5

3. 类的设计……………………………………………… 6

4. 主程序设计…………………………………………… 12

**三、调试分析……………………………………………… 14**

**四、测试结果分析………………………………………… 15**

**五、附录…………………………………………………… 22**

**实验题目和要求**

三、 大整数的运算 【问题描述】密码学分为两类密码：对称密码和非对称密码。对称密码主要用于数据的加/ 解密，而非对称密码则主要用于认证、数字签名等场合。非对称密码在加密和解密时，是把 加密的数据当作一个大的正整数来处理，这样就涉及到大整数的加、减、乘、除和指数运算 等，同时，还需要对大整数进行输出。请采用相应的数据结构实现大整数的加、减、乘、除 和指数运算，以及大整数的输入和输出。

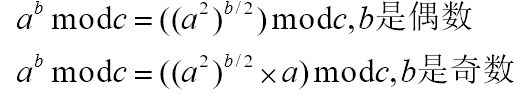
【基本要求】 1. 要求采用链表来实现大整数的存储和运算，不允许使用标准模板类的链表类(list)和函 数。同时要求可以从键盘输入大整数，也可以文件输入大整数，大整数可以输出至显示 器，也可以输出至文件。大整数的存储、运算和显示，可以同时支持二进制和十进制， 但至少要支持十进制。大整数输出显示时，必须能清楚地表达出整数的位数。测试时， 各种情况都需要测试，并附上测试截图；要求测试例子要比较详尽，各种极限情况也要 考虑到，测试的输出信息要详细易懂，表明各个功能的执行正确； 2. 要求大整数的长度可以不受限制，即大整数的十进制位数不受限制，可以为十几位的整 数，也可以为 500 多位的整数，甚至更长；大整数的运算和显示时，只需要考虑正的大 整数。如果可能的话，请以秒为单位显示每次大整数运算的时间； 3. 要求采用类的设计思路，不允许出现类以外的函数定义，但允许友元函数。主函数中只 能出现类的成员函数的调用，不允许出现对其它函数的调用。 4. 要求采用多文件方式：.h 文件存储类的声明，.cpp 文件存储类的实现，主函数 main 存 储在另外一个单独的 cpp 文件中。如果采用类模板，则类的声明和实现都放在.h 文件中。

5. 不强制要求采用类模板，也不要求采用可视化窗口；要求源程序中有相应注释； 6. 建议采用 Visual C++ 6.0 及以上版本进行调试；

【实现提示】 1. 大整数的加减运算可以分解为普通整数的运算来实现；而大整数的乘、除和指数运算， 可以分解为大整数的加减运算。 2. 大整数的加、减、乘、除和指数运算，一般是在求两大整数在取余操作下的加、减、乘、 除和指数运算，即分别求 (a +b) mod n, (a - b) mod n, (a \* b) mod n, (a / b) mod n 和(a ^ b) mod n。其中 a ^ b 是求 a 的 b 次方，而 n 称之为模数。说明：取余操作(即 mod 操作) 是计算相除之后所得的余数，不同于除法运算的是，取余操作得到的是余数，而不是除 数。如 7 mod 5 = 2。模数 n 的设定，可以为 2m 或 10m，m 允许每次计算时从键盘输入。 模数 n 的取值一般为 2512(相当于十进制 150 位左右),21024(相当于十进制 200~300 位),22048(相当于十进制 300~500 位)。为了测试，模数 n 也可以为 2256, 2128等值。 3. 需要设计主要类有：链表类和大整数类。链表类用于处理链表的相关操作，包括缺省构 造函数、拷贝构造函数、赋值函数、析构函数、链表的创建、插入、删除和显示等；而 大整数类则用于处理大整数的各种运算和显示等

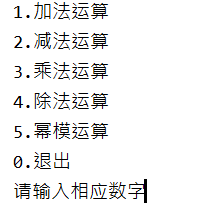
**设计思路**

1. **系统总体设计**
2. 大整数的储存方式按题目要求使用**链表**保存，链表的每一个结点存储一个不大于1000的整数（三位数字），将大整数每三位做一个划分，存储到链表的结点中。
3. 大整数的加法的实现是，从最低位开始，通过两个链表对应位的结点相加，两个结点相加大于1000则产生进位1，并将进位1加到后一个结点中。
4. 大整数的减法实现与加法类似，从最低位开始，两个链表对应位的结点相减，小于0则产生借位1，并在后一个结点中减去借位1。
5. 大整数乘法基于大整数加法，采用分治法实现。第一个大整数的每一个结点与另一个大整数相乘形成一个新的大整数，用已经实现的大整数加法将这些大整数相加，得到大整数乘法结果。
6. 大整数除法基于大整数减法实现，根据被除数和除数相差的位数，每次减去10的n次倍除数，直到被除数被减到小于除数。
7. 大整数幂模运算采用快速幂算法，运用到以下公式用循环削减幂指数，并要借助已经实现的大整数乘法。

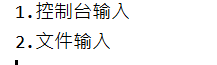


* 1. **系统功能设计**

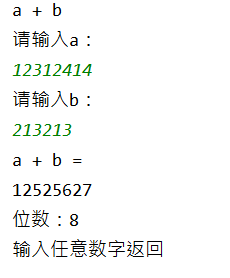
实现大整数的加减乘除和幂模运算。如下图所示



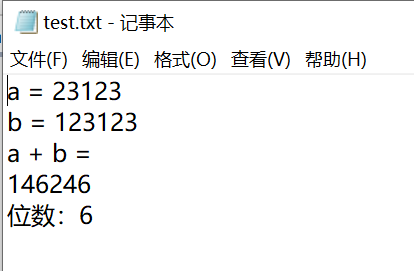
每种运算支持从控制台和文件输入，同时支持输入到控制台或文件。



下图为控制台输入和输出



下图为文件输入和输出



**三、类的设计**

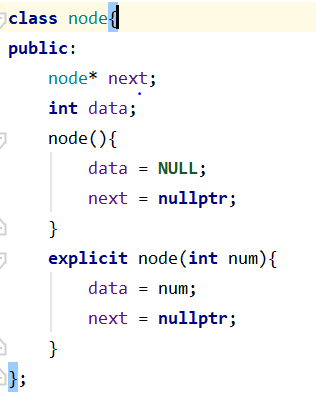
该程序包含三个类。node类（结点），linked\_list类（链表），largeInt类（大整数）。

1. node类

有两个属性，data 和 next。

data用于存储一个不大于1000的整数。

next用于访问下一个结点。

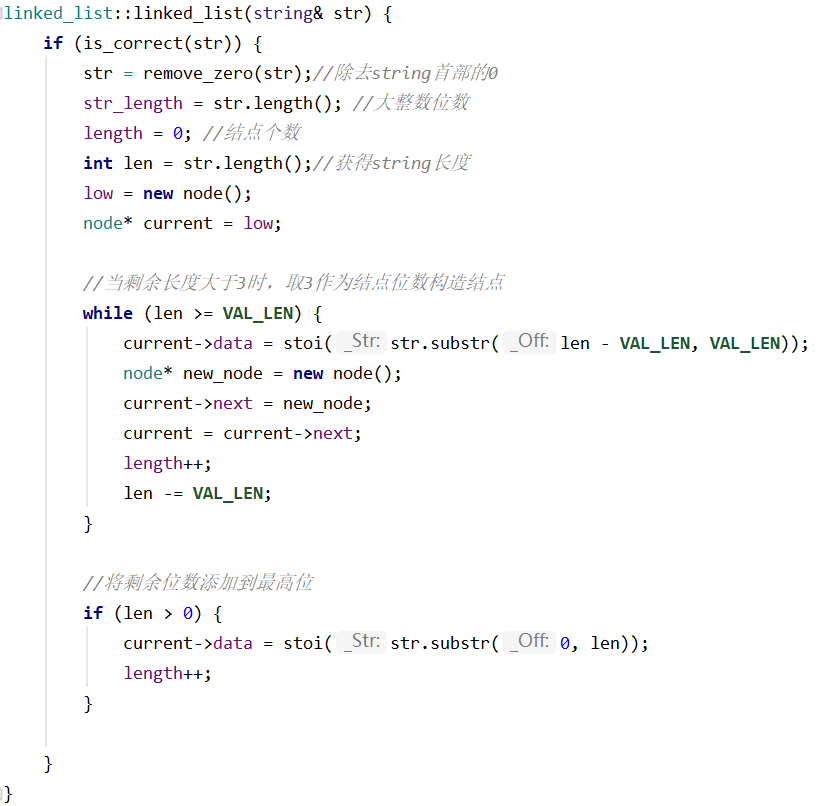


1. linked\_list类

用于存储大整数的相关的信息，并包含一些操作结点的方法。



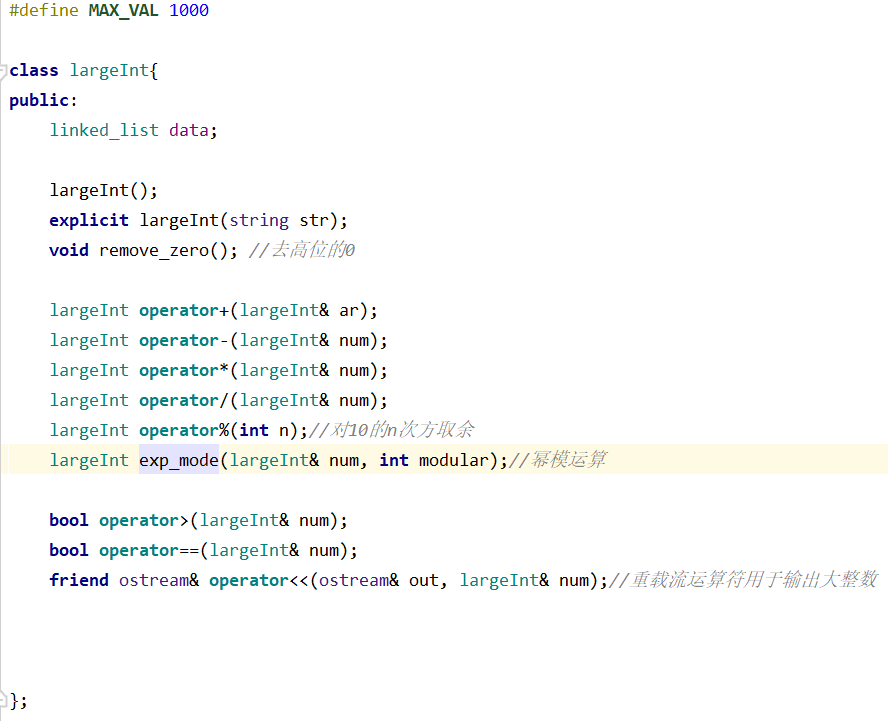
下图为传入string的构造函数，从string中每取三位构造一个结点，并添加到链表中



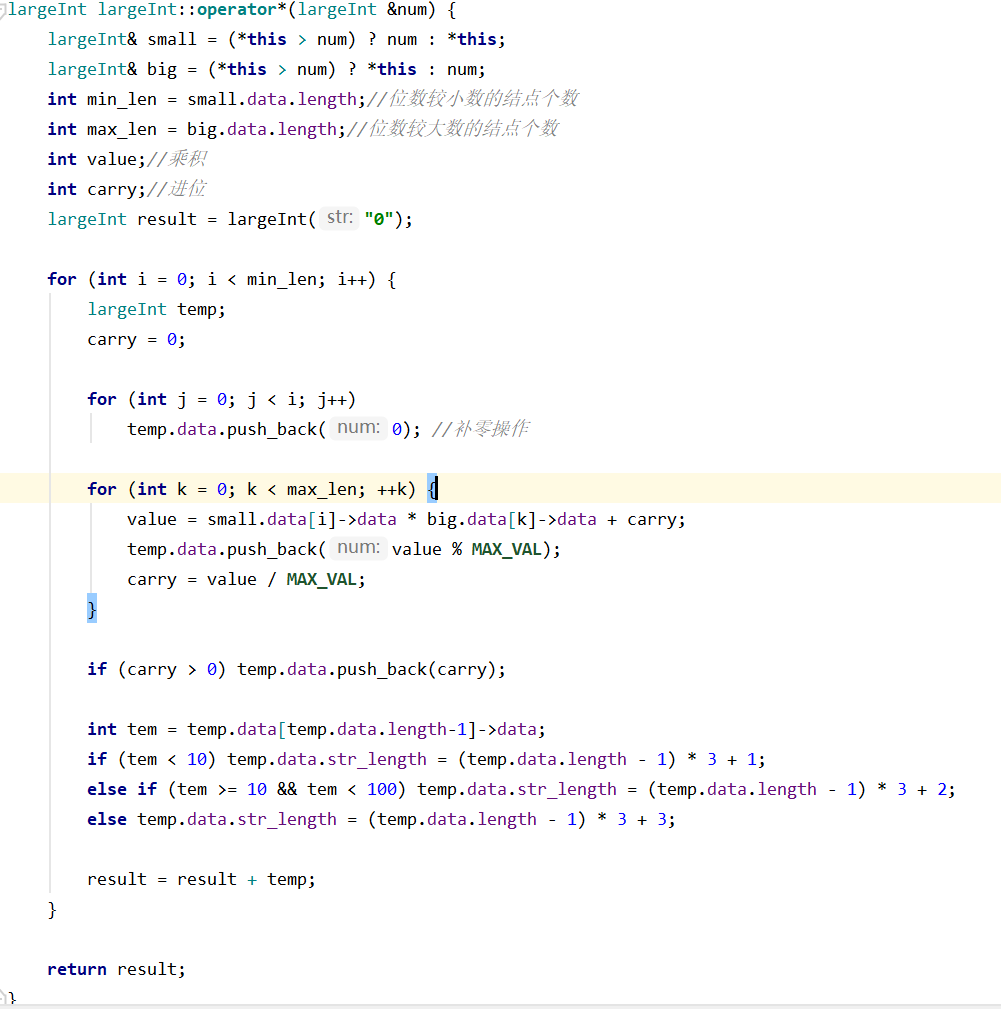
1. largeInt类

该类存储大整数，并重载了“+”、“-”、“\*”、“/”、“%”、“==”、“>”、“<<”运算符用于大整数的运算和输出大整数。

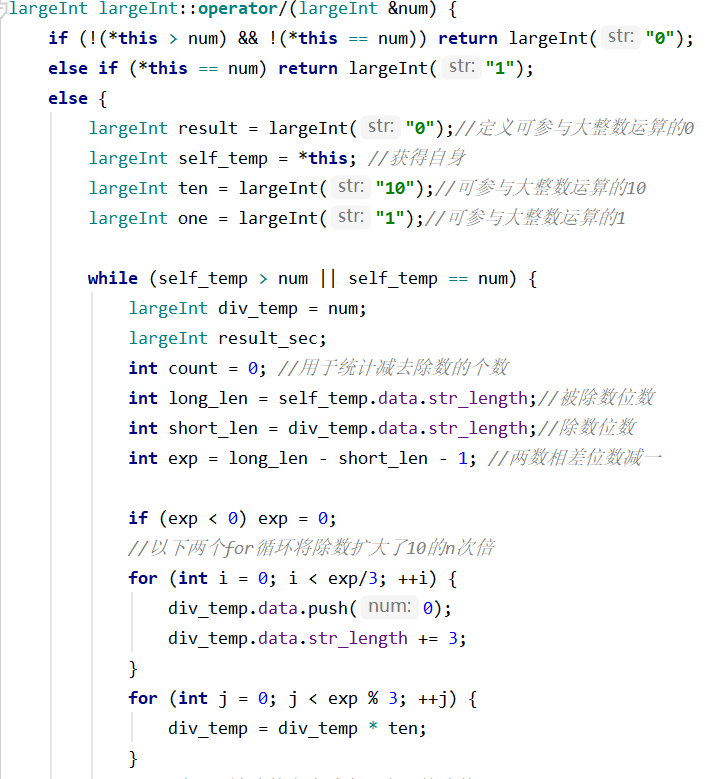
另外定义了exp\_mode方法用于大整数的幂模运算。

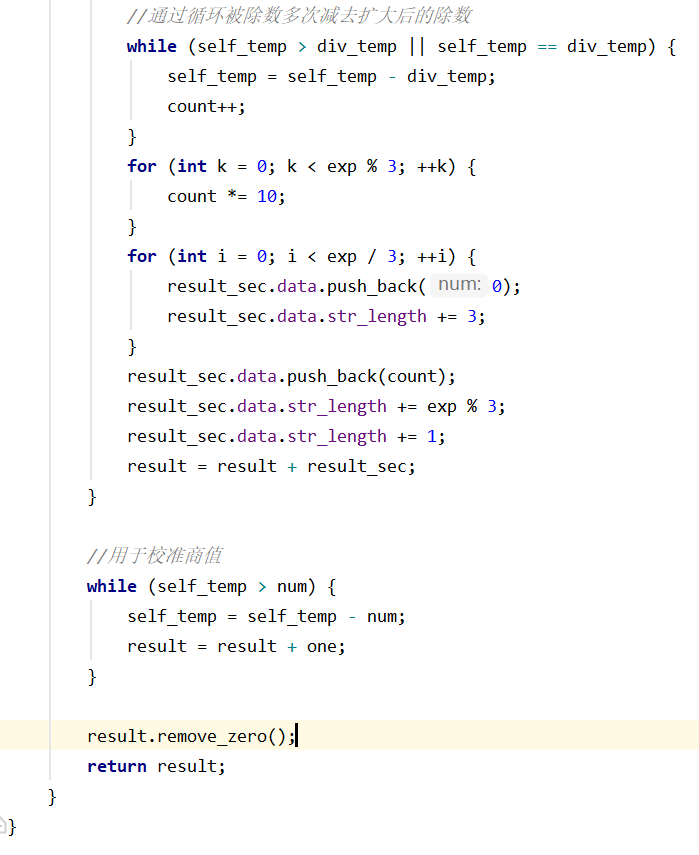


重载\*运算符: 分别取出较小大整数的每一个结点，根据结点位置为临时大整数补上相应数量的0，将较小大整数的每一个结点与另一个大整数的每个结点相乘形成临时大整数，用已经实现的大整数加法将这些临时大整数相加，得到大整数乘法结果。

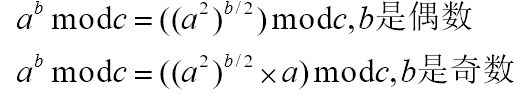


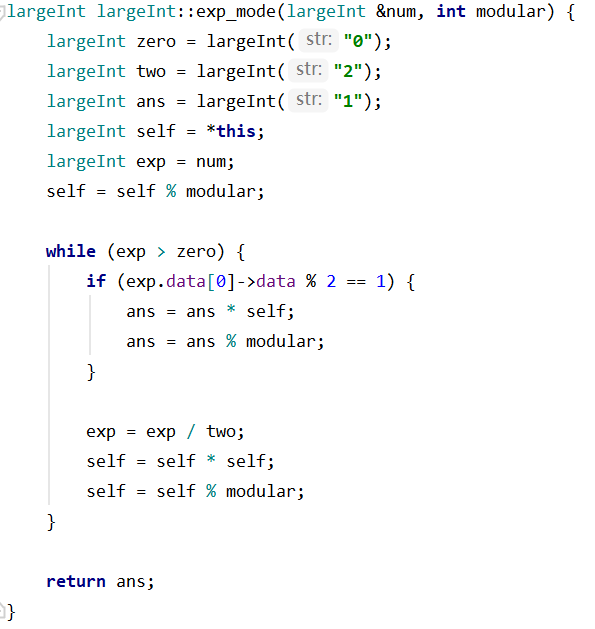
重载/运算符： 先判断除数，除数大于被除数返回0，两数相等则返回1。再讨论除数小于被除数。除法基于减法实现。被除数多次减去10的n次倍除数，直到被除数小于除数。减去的除数个数即为除法的商值。





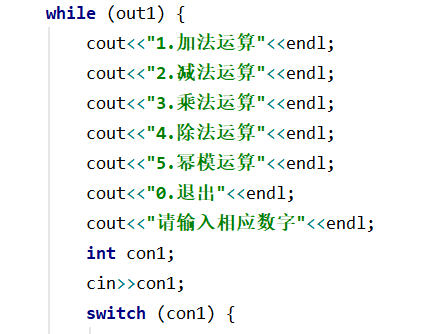
定义幂模运算函数: 大整数幂模运算采用快速幂算法，运用到以下公式用循环削减幂指数，并要借助已经实现的大整数乘法。





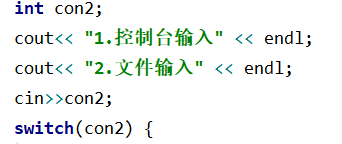
* 1. **主程序设计**

1. 主程序最外层为一个while循环，在条件为true时可多次进行大整数运算。
2. 往内为一个switch语句块，用于选择要进行的运算类型，输入相应数字进行相应的运算，输入0破坏while循环执行条件，退出主程序。



1. 最内层也是一个switch语句块，用于选择输入运算值的方式,

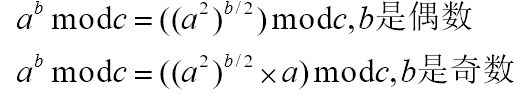
选择控制台输入则结果也会输出到控制台，从文件输入则结果会输出到文件。



**调试分析**

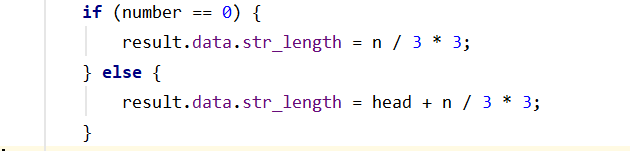
技术难点:

* + 1. 大整数乘法需采用分治法算法
    2. 幂模运算需要用到快速幂算法



调试错误分析:

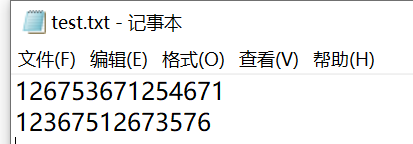
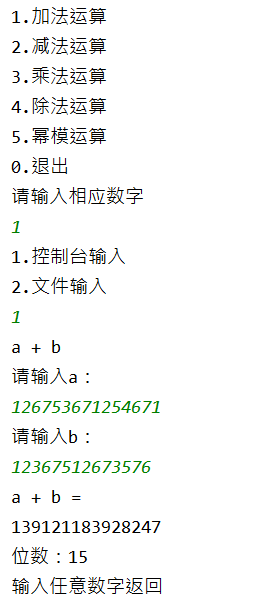
之前程序在进行大整数幂模运算时经常会造成程序崩溃，经调试分析发现，幂模运算中进行大整数乘法时程序发生了崩溃，原因是对大整数进行取余操作时，未对存储大整数位数的属性进行修改，导致大整数无法正常进行乘法运算，从而程序崩溃。在重载的%操作符中添加了如下校正大整数位数的语句后，程序即可正常运行。

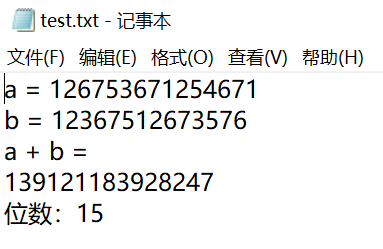
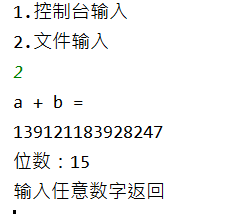


**测试结果分析**

**加法测试**: **a + b**

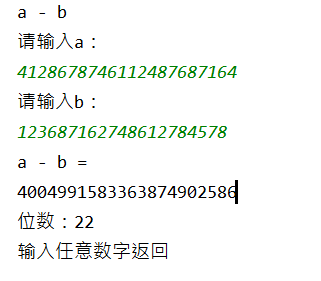
1.a = 126753671254671 b = 12367512673576





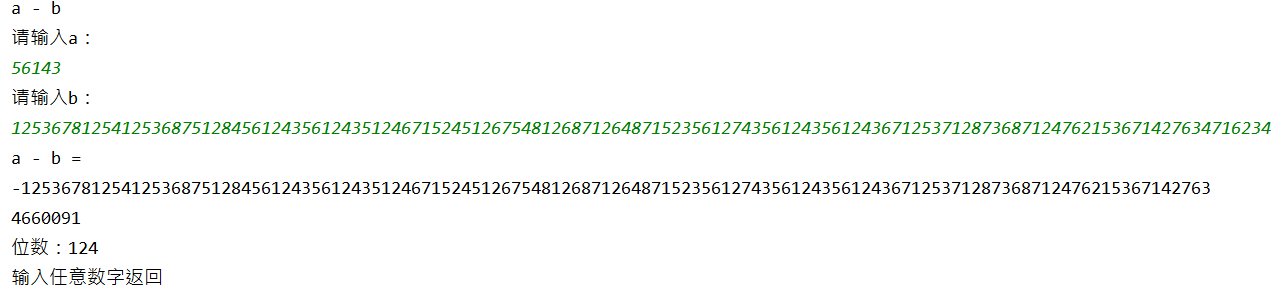
**减法测试:a - b**

1. a = 4128678746112487687164 b = 123687162748612784578



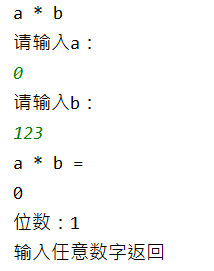
1. a = 56143

b = 125367812541253687512845612435612435124671524512675481268712648715235612743561243561243671253712873687124762153671427634716234

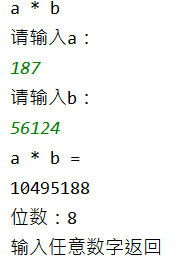


**乘法测试: a \* b**

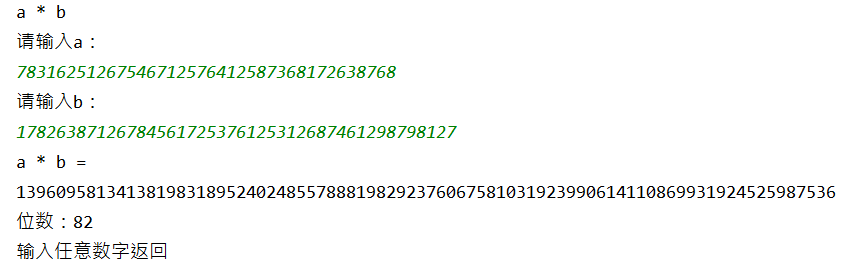
1.



2.

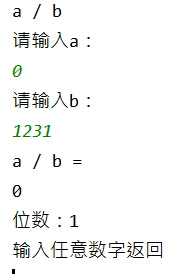


3.

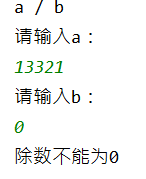


**除法测试：a / b**

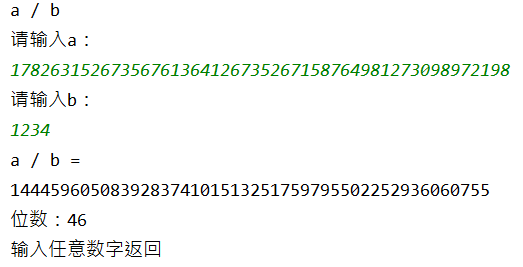
1.



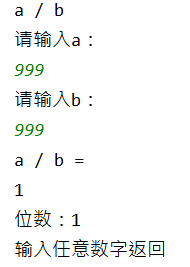
2.



3.

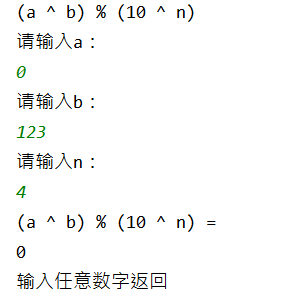


4.

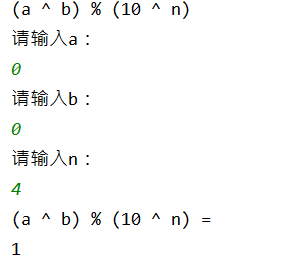


**幂模运算: (a ^ b) / (10 ^ n)**

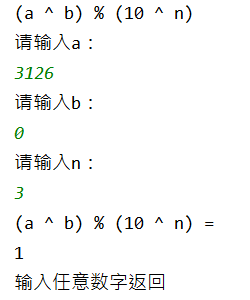
1.



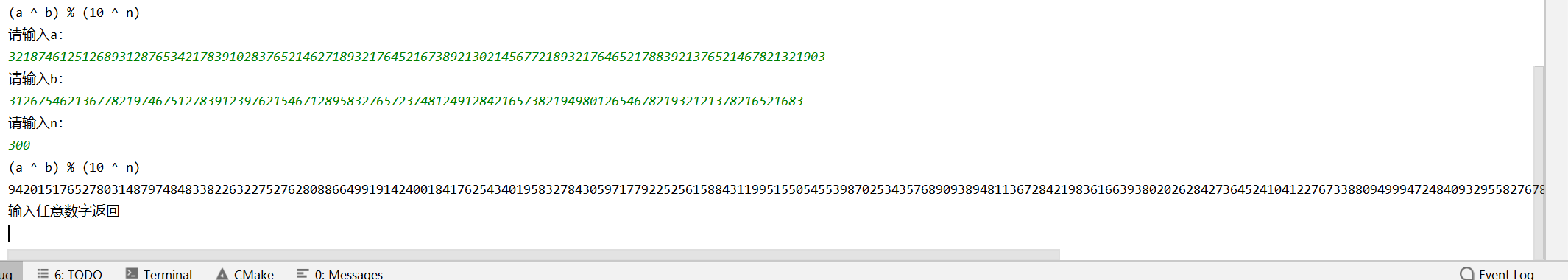
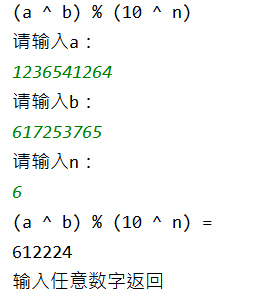
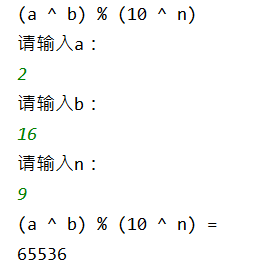
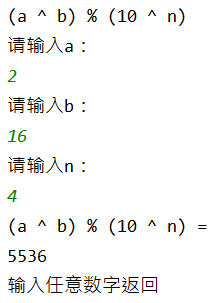
2.



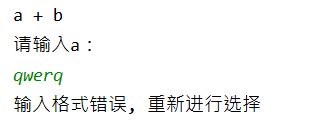
3.



4.



**其他测试：**



**附录**

**node.h**

*//  
// Created by 20180 on 2019/11/19.  
//*#ifndef **C\_NODE\_H**#define **C\_NODE\_H  
  
class** node{  
**public**:  
 node\* next;  
 **int** data;  
 node(){  
 data = **NULL**;  
 next = **nullptr**;  
 }  
 **explicit** node(**int** num){  
 data = num;  
 next = **nullptr**;  
 }  
};  
  
#endif *//C\_NODE\_H*

**linked\_list.h**

*//  
// Created by 20180 on 2019/11/19.  
//*#include **<iostream>**#include **<string>**#include **"node.h"  
using namespace** std;  
  
#ifndef **C\_LINKED\_LIST\_H**#define **C\_LINKED\_LIST\_H**#define **VAL\_LEN** 3 *//每个结点最多存储的位数***class** linked\_list{  
**public**:  
 **int** str\_length; *//位数* **int** length; *//分组数* node\* low; *//表头，存储最低位* linked\_list();*//无参构造* **explicit** linked\_list(string& str);*//传入string类型的构造* **void** push(**int** num);*//插入低位* **static bool** is\_correct(**const** string& str); *//判断是否输入正确的大整数* **void** push\_back(**int** num); *//插入高位* **void** pop\_back(); *//用于除去高位的0，位数没有同步减去* **static** string remove\_zero(string str);*//去首部的零* node\* **operator**[](**int** index);*//重载[]运算符，便于访问index处的结点*};  
#endif *//C\_LINKED\_LIST\_H*

**Linked\_list.cpp**

*//  
// Created by 20180 on 2019/11/20.  
//*#include **"linked\_list.h"**#include **<utility>  
  
bool** linked\_list::is\_correct(**const** string& str) {  
 **for** (**char** i : str) {  
 **if** (i >= **'0'** && i <= **'9'**)  
 **continue**;  
 **return false**;  
 }  
 **return true**;  
}  
  
linked\_list::linked\_list() {  
 length = 0;  
 str\_length = 0;  
 low = **new** node();  
}  
  
linked\_list::linked\_list(string& str) {  
 **if** (is\_correct(str)) {  
 str = remove\_zero(str);*//除去string首部的0* str\_length = str.length(); *//大整数位数* length = 0; *//结点个数* **int** len = str.length();*//获得string长度* low = **new** node();  
 node\* current = low;  
  
 *//当剩余长度大于3时，取3作为结点位数构造结点* **while** (len >= **VAL\_LEN**) {  
 current->data = stoi(str.substr(len - **VAL\_LEN**, **VAL\_LEN**));  
 node\* new\_node = **new** node();  
 current->next = new\_node;  
 current = current->next;  
 length++;  
 len -= **VAL\_LEN**;  
 }  
  
 *//将剩余位数添加到最高位* **if** (len > 0) {  
 current->data = stoi(str.substr(0, len));  
 length++;  
 }  
  
 }  
}  
*// 该函数无bug*string linked\_list::remove\_zero(string str) {  
 string new\_str = std::move(str);  
 **if** (new\_str.length() > 1) {  
 **bool** ok = **false**;  
 **int** index = 0;  
 **while** (!ok) {  
 **if** (new\_str[index] != **'0'**)  
 ok = **true**;  
 **else** {  
 new\_str.erase(index,1);  
 }  
 }  
 }  
  
 **return** new\_str;  
}  
  
node\* linked\_list::**operator**[](**int** index) {  
 node\* res = low;  
 **for** (**int** i = 0; i < index; ++i) {  
 res = res->next;  
 }  
 **return** res;  
}  
  
**void** linked\_list::push\_back(**int** num) {  
 **if** (length == 0) {  
 low->data = num;  
 length++;  
 } **else** {  
 node\* current = low;  
 **for** (**int** i = 1; i < length; ++i) {  
 current = current->next;  
 }  
  
 node\* new\_node = **new** node(num);  
 current->next = new\_node;  
 length++;  
 }  
  
}  
  
**void** linked\_list::pop\_back() {  
 **delete this**->**operator**[](length - 1);  
 **this**->**operator**[](length - 2)->next = **nullptr**;  
 length--;  
}  
  
**void** linked\_list::push(**int** num) {  
 **if** (length == 0) {  
 low->data = num;  
 length++;  
 } **else** {  
 node\* new\_node = **new** node(num);  
 new\_node->next = low;  
 low = new\_node;  
 length++;  
 }  
}

**large\_int.h**

*//  
// Created by 20180 on 2019/11/20.  
//*#include **"linked\_list.h"**#ifndef **C\_LARGE\_INT\_H**#define **C\_LARGE\_INT\_H**#define **MAX\_VAL** 1000  
  
**class** largeInt{  
**public**:  
 linked\_list data;  
  
 largeInt();  
 **explicit** largeInt(string str);  
 **void** remove\_zero(); *//去高位的0* largeInt **operator**+(largeInt& ar);  
 largeInt **operator**-(largeInt& num);  
 largeInt **operator**\*(largeInt& num);  
 largeInt **operator**/(largeInt& num);  
 largeInt **operator**%(**int** n);*//对10的n次方取余* largeInt exp\_mode(largeInt& num, **int** modular);*//幂模运算* **bool operator**>(largeInt& num);  
 **bool operator**==(largeInt& num);  
 **friend** ostream& **operator**<<(ostream& out, largeInt& num);*//重载流运算符用于输出大整数*};  
#endif *//C\_LARGE\_INT\_H*

**large\_int.cpp**

*//  
// Created by 20180 on 2019/11/20.  
//*#include **"large\_int.h"**largeInt::largeInt(string str) {  
 data = linked\_list(str);  
}  
  
largeInt::largeInt() {  
 data = linked\_list();  
}  
  
**void** largeInt::remove\_zero() {  
 **int** index = data.length;  
 **while** (--index >= 1) {  
 **if** (data[index]->data > 0) **break**;  
  
 data.pop\_back();  
 }  
}  
  
ostream& **operator**<<(ostream &out, largeInt &num) {  
 out<<num.data[num.data.length-1]->data;  
 **for** (**int** i = num.data.length-2; i >= 0; i--) {  
 **int** number = num.data[i]->data;  
 **if** (number / 10 == 0) {  
 out << **"00"** << number;  
 } **else if** (number / 100 == 0) {  
 out << **"0"** << number;  
 } **else** {  
 out << number;  
 }  
 }  
 **return** out;  
}  
  
**bool** largeInt::**operator**>(largeInt &num) {  
 **bool** result = **true**;  
  
 **if** (data.str\_length < num.data.str\_length)  
 result = **false**;  
 **else if** (data.str\_length == num.data.str\_length) {  
 **for** (**int** i = data.length - 1; i >= 0; i--) {  
 **if** (data[i]->data < num.data[i]->data) {  
 result = **false**;  
 **break**;  
 } **else if** (data[i]->data > num.data[i]->data) {  
 **break**;  
 } **else**{  
  
 }  
 }  
 }  
  
 **if** (\***this** == num) result = **false**;  
  
 **return** result;  
}  
  
**bool** largeInt::**operator**==(largeInt &num) {  
 **bool** eq = **true**;  
 **if** (data.str\_length != num.data.str\_length) {  
 eq = **false**;  
 **return** eq;  
 } **else** {  
 **for** (**int** i = 0; i < data.length; ++i) {  
 **if** (data[i]->data != num.data[i]->data) {  
 eq = **false**;  
 **break**;  
 }  
 }  
 **return** eq;  
 }  
}  
  
largeInt largeInt::**operator**+(largeInt &ar) {  
 **int** carry = 0; *//进位，0或1* **int** min\_len = (\***this** > ar) ? ar.data.length : data.length;  
 **int** max\_len = (\***this** > ar) ? data.length : ar.data.length;  
 largeInt& extra = (\***this** > ar) ? (\***this**) : ar; *//取位数较多的大整数* largeInt result = largeInt();  
 **int** value = 0;  
  
 **for** (**int** i = 0; i < min\_len; ++i) {  
 value = data[i]->data + ar.data[i]->data + carry;  
  
 **if** (value >= **MAX\_VAL**) {  
 carry = 1;  
 result.data.push\_back(value - **MAX\_VAL**);  
 } **else** {  
 carry = 0;  
 result.data.push\_back(value);  
 }  
 }  
  
 **for** (**int** j = min\_len; j < max\_len; ++j) {  
 value = extra.data[j]->data + carry;  
  
 **if** (value >= **MAX\_VAL**) {  
 result.data.push\_back(value-**MAX\_VAL**);  
 carry = 1;  
 } **else** {  
 result.data.push\_back(value);  
 carry = 0;  
 }  
 }  
  
 result.data.str\_length = extra.data.str\_length;  
  
 **if** (carry > 0) {  
 result.data.str\_length += 1;  
 result.data.push\_back(carry);  
 }  
  
 **return** result;  
}  
  
largeInt largeInt::**operator**-(largeInt &num) {  
 **int** min\_len = (\***this** > num) ? num.data.length : data.length;  
 **int** max\_len = (\***this** > num) ? data.length : num.data.length;  
 largeInt& big = (\***this** > num) ? (\***this**) : num; *//取位数较多的大整数* largeInt& small = (\***this** > num) ? num : \***this**;  
 largeInt result = largeInt();  
 **int** value = 0; *//差* **int** carry = 0; *//借位* **for** (**int** i = 0; i < min\_len; ++i) {  
 **if** (big.data[i]->data < small.data[i]->data + carry) {  
 value = big.data[i]->data + **MAX\_VAL** - small.data[i]->data - carry;  
 carry = 1;  
 } **else** {  
 value = big.data[i]->data - small.data[i]->data - carry;  
 carry = 0;  
 }  
  
 result.data.push\_back(value);  
 }  
  
 **for** (**int** j = min\_len; j < max\_len; ++j) {  
 **if** (big.data[j]->data < carry) {  
 value = big.data[j]->data + **MAX\_VAL** - carry;  
 carry = 1;  
 } **else** {  
 value = big.data[j]->data - carry;  
 carry = 0;  
 }  
  
 result.data.push\_back(value);  
 }  
  
 result.remove\_zero();  
  
 **if** (!(\***this** > num)) {  
 result.data[result.data.length-1]->data = -result.data[result.data.length-1]->data;  
 }  
  
 **int** tem = result.data[result.data.length - 1]->data;  
 **if** (tem < 10) result.data.str\_length = (result.data.length - 1) \* 3 + 1;  
 **else if** (tem >= 10 && tem < 100) result.data.str\_length = (result.data.length - 1) \* 3 + 2;  
 **else** result.data.str\_length = (result.data.length - 1) \* 3 + 3;  
  
 **return** result;  
}  
  
largeInt largeInt::**operator**\*(largeInt &num) {  
 largeInt& small = (\***this** > num) ? num : \***this**;  
 largeInt& big = (\***this** > num) ? \***this** : num;  
 **int** min\_len = small.data.length;*//位数较小数的结点个数* **int** max\_len = big.data.length;*//位数较大数的结点个数* **int** value;*//乘积* **int** carry;*//进位* largeInt result = largeInt(**"0"**);  
  
 **for** (**int** i = 0; i < min\_len; i++) {  
 largeInt temp;  
 carry = 0;  
  
 **for** (**int** j = 0; j < i; j++)  
 temp.data.push\_back(0); *//补零操作* **for** (**int** k = 0; k < max\_len; ++k) {  
 value = small.data[i]->data \* big.data[k]->data + carry;  
 temp.data.push\_back(value % **MAX\_VAL**);  
 carry = value / **MAX\_VAL**;  
 }  
  
 **if** (carry > 0) temp.data.push\_back(carry);  
  
 **int** tem = temp.data[temp.data.length-1]->data;  
 **if** (tem < 10) temp.data.str\_length = (temp.data.length - 1) \* 3 + 1;  
 **else if** (tem >= 10 && tem < 100) temp.data.str\_length = (temp.data.length - 1) \* 3 + 2;  
 **else** temp.data.str\_length = (temp.data.length - 1) \* 3 + 3;  
  
 result = result + temp;  
 }  
  
 **return** result;  
}  
  
largeInt largeInt::**operator**/(largeInt &num) {  
 **if** (!(\***this** > num) && !(\***this** == num)) **return** largeInt(**"0"**);  
 **else if** (\***this** == num) **return** largeInt(**"1"**);  
 **else** {  
 largeInt result = largeInt(**"0"**);*//定义可参与大整数运算的0* largeInt self\_temp = \***this**; *//获得自身* largeInt ten = largeInt(**"10"**);*//可参与大整数运算的10* largeInt one = largeInt(**"1"**);*//可参与大整数运算的1* **while** (self\_temp > num || self\_temp == num) {  
 largeInt div\_temp = num;  
 largeInt result\_sec;  
 **int** count = 0; *//用于统计减去除数的个数* **int** long\_len = self\_temp.data.str\_length;*//被除数位数* **int** short\_len = div\_temp.data.str\_length;*//除数位数* **int** exp = long\_len - short\_len - 1; *//两数相差位数减一* **if** (exp < 0) exp = 0;  
 *//以下两个for循环将除数扩大了10的n次倍* **for** (**int** i = 0; i < exp/3; ++i) {  
 div\_temp.data.push(0);  
 div\_temp.data.str\_length += 3;  
 }  
 **for** (**int** j = 0; j < exp % 3; ++j) {  
 div\_temp = div\_temp \* ten;  
 }  
 *//通过循环被除数多次减去扩大后的除数* **while** (self\_temp > div\_temp || self\_temp == div\_temp) {  
 self\_temp = self\_temp - div\_temp;  
 count++;  
 }  
 **for** (**int** k = 0; k < exp % 3; ++k) {  
 count \*= 10;  
 }  
 **for** (**int** i = 0; i < exp / 3; ++i) {  
 result\_sec.data.push\_back(0);  
 result\_sec.data.str\_length += 3;  
 }  
 result\_sec.data.push\_back(count);  
 result\_sec.data.str\_length += exp % 3;  
 result\_sec.data.str\_length += 1;  
 result = result + result\_sec;  
 }  
  
 *//用于校准商值* **while** (self\_temp > num) {  
 self\_temp = self\_temp - num;  
 result = result + one;  
 }  
  
 result.remove\_zero();  
 **return** result;  
 }  
}  
  
largeInt largeInt::**operator**%(**int** n) {  
 **if** (data.str\_length < n) {  
 **return** \***this**;  
 } **else** {  
 **int** index = n / 3;  
 largeInt result;  
  
 **for** (**int** i = 0; i < index; ++i) {  
 result.data.push\_back(data[i]->data);  
 }  
  
 **int** head = n % 3;  
 **int** number = 0;  
  
 **switch** (head){  
 **case** 0: number = 0;  
 **break**;  
 **case** 1: number = data[index]->data % 10;  
 **break**;  
 **case** 2: number = data[index]->data % 100;  
 **break**;  
 **default**: ;  
 }  
 result.data.push\_back(number);  
 result.remove\_zero();  
 **if** (number == 0) {  
 result.data.str\_length = n / 3 \* 3;  
 } **else** {  
 result.data.str\_length = head + n / 3 \* 3;  
 }  
  
 **return** result;  
 }  
}  
  
largeInt largeInt::exp\_mode(largeInt &num, **int** modular) {  
 largeInt zero = largeInt(**"0"**);  
 largeInt two = largeInt(**"2"**);  
 largeInt ans = largeInt(**"1"**);  
 largeInt self = \***this**;  
 largeInt exp = num;  
 self = self % modular;  
  
 **while** (exp > zero) {  
 **if** (exp.data[0]->data % 2 == 1) {  
 ans = ans \* self;  
 ans = ans % modular;  
 }  
  
 exp = exp / two;  
 self = self \* self;  
 self = self % modular;  
 }  
  
 **return** ans;  
}