

数据结构课程设计

大整数的运算

实验报告





学生姓名：郑东晖

专业班级：大数据工程1901

学生学号：201906062631

完成日期：2021/1/18

目录

**一、实验题目1**

**二、实验要求1**

**三、设计思路2**

（1）系统整体设计2

（2）系统功能设计4

（3）类的设计4

node类5

linkList类5

bigInteger类7

action类20

（4）主程序的设计21

**四、调试分析22**

（1）技术难点分析22

（2）调试错误分析23

**五、测试结果分析24**

（1）不涉及文件输入操作25

十进制25

加法25

减法25

指数运算26

乘法27

除法27

二进制28

加法28

减法28

指数运算29

乘法31

除法31

（2）涉及文件输入操作31

十进制31

加法32

减法32

乘法33

除法33

指数运算33

二进制34

加法34

减法35

乘法35

除法35

指数运算36

（3）退出程序操作37

**六、附录38**

Node.h38

LinkList.h39

LinkList.cpp43

BigInteger.h50

BigInteger.cpp57

Action.h103

Action.vpp104

Main.cpp115

# 一、实验题目

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。

# 二、实验要求

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。

# 三、设计思路

## (1)系统整体设计

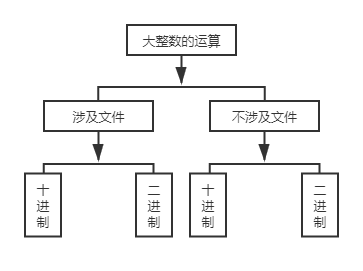


图1 系统整体功能示意图

大整数运算（Operation of large integers：OLI）系统，整体框架示意图如图1所示。数据获取主要有两个渠道，即不涉及文件的键盘输入和涉及文件的文件流输入，支持十进制和二进制两种数据方式，并且支持加法、减法、乘法、除法和指数运算五种运算。十进制的加法、减法、乘法和除法支持负数的运算。

大整数的存储采用双向链表的数据结构（自定义类），大整数的运算在bigInteger类中进行实现，包括大整数的加法、减法、乘法、除法和指数运算（包括十进制和二进制）。系统的主程序设计（对于大整数运算的各个函数的调用，各功能模块的调用实现）在action类中实现。系统的使用从main()函数开始，将通过调用action类中的welcomeScreen()函数开始。

该系统共包含四个类：

表1

|  |
| --- |
| action（操作类） |
| bigInteger（大整数类） |
| linkList（双向链表类） |
| node（节点类） |

工程包含三种文件类型（.txt,.h,.cpp），共计12个文件：

.txt：

表2

|  |  |
| --- | --- |
| file\_decimalSystem.txt | 十进制数据输入文件 |
| file\_binarySystem.txt | 二进制数据输入文件 |
| result\_decimalSystem.txt | 十进制数据运算结果输出文件 |
| result\_binarySystem.txt | 二进制数据运算结果输出文件 |

.h

表3

|  |  |
| --- | --- |
| Action.h | 操作类的声明 |
| BigInteger.h | 大整数类的声明 |
| LinkList.h | 双向链表类的声明 |
| Node.h | 节点类的声明与实现 |

.cpp

表4

|  |  |
| --- | --- |
| Action.cpp | 操作类的实现 |
| BigInteger.cpp | 大整数类的实现 |
| LinkList.cpp | 双向链表类的实现 |
| main.cpp | 系统函数入口 |

## （2）系统功能设计

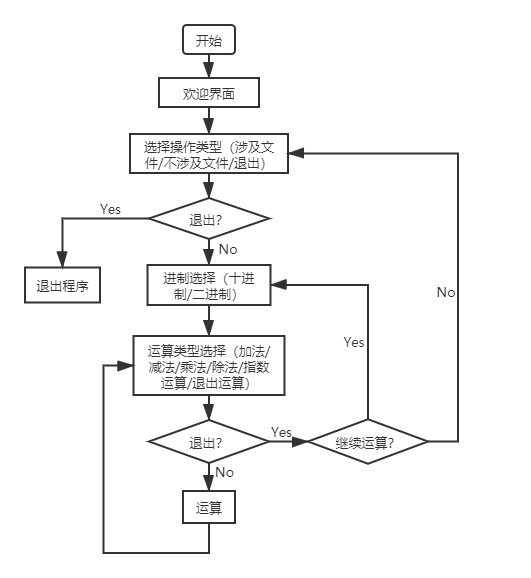


图2 总体功能流程图

该运算系统的总体功能流程如图2所示：当程序开始运行后，调出欢迎界面，按任意键后进入操作类型选择界面，选择是否涉及文件操作或者退出程序。若选择Yes，则直接退出程序；若选择No，则依据选择（不涉及文件/涉及文件）进入相应接口。然后调用进制选择界面，之后调出运算类型选择页面，若选择五种运算中的其中一种，则进入运算，依照提示进行键盘输入或者文件流输入大整数进行运算，运算结束后再次调用运算类型选择页面，以此循环；若选择退出，退出后则提示，是否继续运算，如果是，则进入进制选择界面，如果否，则回到操作类型选择界面。

## （3）类的设计

以下详细解释每一个类的成员和成员函数：

表5 node类总览

|  |  |  |
| --- | --- | --- |
| node类 | | |
| 功能 | 用于双向链表的节点，存储数据 | |
| public成员变量 | | |
| 变量名 | 变量类型 | 变量描述 |
| data | int | 存储的数据 |
| prev | node\* | 前驱指针 |
| next | node\* | 后继指针 |
| 成员函数 | | |
| node()  无参构造函数 | | |
| explicit node(const int &item, node \*prevNode = nullptr, node \*nextNode = nullptr)  有参构造函数 | | |

Node类的无参构造函数，默认将data赋值为0，pre和next指针都赋值为nullptr；有参构造函数，data、pre指针、next指针赋值为传入的数据item、preNode、nextNode；两个指针若无传入参数，则默认赋值为nullptr，表示为孤立节点。

表6 linkList类

|  |  |  |
| --- | --- | --- |
| linkList类 | | |
| 功能 | 用于存储大整数，并完成与双向链表相关的操作 | |
| private成员变量 | | |
| 变量名 | 变量类型 | 变量描述 |
| headPoint | node\* | 头指针 |
| linkListLength | int | 链表长度 |
| flag | int | 大整数正负标记 |
| 成员函数 | | |
| node\* returnHeadPoint()  返回头指针 | | |
| int& returnLinkListLength()  返回链表长度的引用 | | |
| int& returnLinkListFlag()  返回大整数符号标记的引用 | | |
| int linkListSize() const  返回链表长度 | | |
| int linkListFlag() const  返回大整数符号标记 | | |
| linkList()  无参构造函数 | | |
| linkList(const linkList &object)  拷贝构造函数 | | |
| linkList & operator=(const linkList &rightHandObject)  赋值运算符重载 | | |
| ~linkList()  析构函数 | | |
| void display(ostream &out)  输出函数，用于输出链表 | | |
| void createLinkListWithoutForValue(int length)  创建长度为length的双向链表,节点data不进行赋值 | | |
| void createLinkListWithForValue(int length)  创建长度为length的双向链表,节点data进行赋值为0 | | |
| void insert(int item, int position)  在position位置后面插入节点 | | |
| void erase(int position)  删除position位置的节点 | | |
| 类外函数 | | |
| ostream & operator<<(ostream &, linkList &)  输出运算符重载 | | |

returnHeadPoint、returnLinkListLength、returnLinkListFlag、linkListSize、linkListFlag五个函数的实现直接return相应的成员即可；无参构造函数则创建一个孤立头节点，并将前后指针皆指向自己，链表长度为0，符号标记为0，再将头指针赋值为头节点的地址，说明本系统中头指针的指向和头节点的地址是相同的。如果flag的值为-1，则表示链表存储的大整数为负数，如果为1则为正数；拷贝构造函数和赋值运算符重载则是为了防止浅拷贝导致析构时的多次析构，不断创建新节点和遍历原链表，将数据进行复制；析构函数是从头结点开始，依次遍历并且delete遍历到的节点，释放内存空间；display函数则是输出函数，从头节点开始遍历输出每一位数字，在输出前，会进行判断是否是负数；createLinkListWithoutForValue函数和createLinkListWithForValue函数皆是创建一个含多个节点的数据为空的链表框架，后者data属性皆赋值为0，该大整数表示为0；insert函数和erase函数分别为在指定位置插入节点和删除节点；输出运算符重载调用类内输出函数，用于“<<”运算符的使用。

表7 bigInteger类

|  |  |
| --- | --- |
| bigInteger类 | |
| 功能 | 实现基于linkList类的大整数运算操作 |
| 静态成员函数 | |
| static string translateToString(linkList &list)  将大整数（链表）转换为字符串 | |
| static void writeFile(const string& data, const string& url)  将数据写入文件 | |
| static int comparisonOfSize(linkList &list1, linkList &list2)  大整数大小比较 | |
| static bool compareListLength(linkList &list1, linkList &list2)  链表长度比较 | |
| static void inputListWithoutFile(linkList &object1, linkList &object2)  不涉及文件的输入函数,将操作数存入链表 | |
| static void listOperateWithFile\_decimalSystem(linkList &object1, linkList &object2, char choice)  涉及文件的操作函数,十进制 | |
| static void listOperateWithFile\_binarySystem(linkList &object1, linkList &object2, char choice)  涉及文件的操作函数,二进制 | |
| static linkList additionAuxiliaryFunction(linkList &object1, linkList &object2, int base)  加法辅助函数 | |
| static linkList subtractionAuxiliaryFunction(linkList &object1, linkList &object2, int base)  减法辅助函数 | |
| static linkList multiplicationAuxiliaryFunction(linkList &object1, linkList &object2, int base)  乘法辅助函数 | |
| static linkList divisionFunction(linkList &operator1, linkList &operator2, int base)  除法子函数 | |
| static linkList divisionAuxiliaryFunction(linkList &object1, linkList &object2, int base)  除法辅助函数 | |
| static void addition\_bigInteger(linkList &operation1, linkList &operation2, int base)  十进制加法函数,可正可负 | |
| static void addition\_bigInteger\_binarySystem(linkList &object1, linkList &object2)  二进制加法函数 | |
| static void subtraction\_bigInteger(linkList &operation1, linkList &operation2, int base)  十进制减法函数,可正可负 | |
| static void subtraction\_bigInteger\_binarySystem(linkList &object1, linkList &object2)  二进制减法函数 | |
| static void multiplication\_bigInteger(linkList &operation1, linkList &operation2, int base)  十进制乘法函数,可正可负 | |
| static void multiplication\_bigInteger\_binarySystem(linkList &object1, linkList &object2)  二进制乘法函数 | |
| static void division\_bigInteger(linkList &operation1, linkList &operation2, int base)  十进制除法函数,可正可负 | |
| static void division\_bigInteger\_binarySystem(linkList &object1, linkList &object2)  二进制除法函数 | |
| static void exponentialCalculation\_bigInteger(linkList &operation1, linkList &operation2, int base)  十、二进制指数运算函数 | |

translateToString函数用于将一个链表转换为一个字符串，因为链表存储一个大整数的形式是：一个节点存储大整数的一位数，所以，链表是不能轻易存进文件中的，因此在存进文件之前需要将一个链表的n个节点转换为一个字符串存进文件。

相关实现代码如下：

string result(length, 0);//创建长度为l,初始化为0的字符串.

back = list.returnHeadPoint()->prev;

for (int i = length - 1; i >= 0; i--) {//循环将链表数据输入到字符串中.

result[i] = back->data + '0';

back = back->prev;

}

if (list.linkListFlag() == -1) {

result = "-" + result;

}

writeFile函数内，开启文件流，调用translateToString函数后将字符串存进文件。

相关实现代码如下：

void bigInteger::writeFile(const string& data, const string& url) {

ofstream fileStream;//文件流对象.

fileStream.open(url, ios::out | ios::app);

//文件不存在.

·······

fileStream << data << endl;

fileStream.close();

}

comparisonOfSize函数和compareListLength函数都是用于比较，但是后者仅仅比较位数多少，即比较绝对值；前者比较的时候会考虑上flag即符号的正负，比较的是真值；inputListWithoutFile(linkList &object1, linkList &object2)函数是不涉及文件的输入函数，两个参数是两个引用，因此此处是地址传递，将外部定义的两个linkList类对象以地址的形式传进函数进行输入保存数据。

相关实现代码如下：

length = string1.length();

//创建长度为length1的双向循环链表,赋值给object1.

object1.createLinkListWithoutForValue(length);

back = object1.returnHeadPoint()->prev;

object1.returnLinkListFlag() = flag;

listOperateWithFile\_decimalSystem函数和listOperateWithFile\_binarySystem函数分别是涉及文件的十进制和二进制输入、运算操作，因此参数除了两个值传递的链表对象，还有一个代表运算类型的char型参数choice（‘+’‘-’‘\*’‘/’‘^’）。

相关实现代码如下：

case '+': {

cout << "加法 —>>> (list1 + list2) = ";

addition\_bigInteger(object1, object2, 10);//十进制

addition\_bigInteger\_binarySystem(object1, object2);//二进制

break;

}

case '-': {

cout << "减法 —>>> (list1 - list2) = ";

subtraction\_bigInteger(object1, object2, 10);//十进制

subtraction\_bigInteger\_binarySystem(object1, object2);//二进制

break;

}

case '\*': {

cout << "乘法 —>>> (list1 \* list2) = ";

multiplication\_bigInteger(object1, object2, 10);//十进制

multiplication\_bigInteger\_binarySystem(object1, object2);//二进制

break;

}

case '/': {

cout << "除法 —>>> (list1 / list2) = ";

division\_bigInteger(object1, object2, 10);//十进制

division\_bigInteger\_binarySystem(object1, object2);//二进制

break;

}

default: {

cout << "指数运算 —>>> (list1 ^ list2) = ";

exponentialCalculation\_bigInteger(object1, object2, 10);//十进制

exponentialCalculation\_bigInteger(object1, object2, 2);//二进制

break;

}

下面对五种运算做详细的解释：

①加法

additionAuxiliaryFunction——加法辅助函数

addition\_bigInteger——加法运算函数

addition\_bigInteger\_binarySystem——用于二进制的加法运算函数

加法的实现，是通过列竖式的思路实现的，逐位相加。

对于10进制，每位相加大于10则进位。

对于2进制，每位相加对于1则进位。

设计两次循环，并非两重循环。第一次循环，循环位数少的大整数位次，第二次循环循环差位次。在此过程中主要设计了三个变量用于记录循环过程中的各个数据。有sum记录每位相加的和，valueForBit记录对和取余的结果, quotientRecord记录相加后的进位数。设计思想和整数的加法一致，即为满10进位。

相关实现代码如下：

//sum记录每位的和.

int sum = tempPoint1->data + tempPoint2->data + quotientRecord;

int valueForBit = sum % base;//valueForBit为每位应当记录的的结果.

quotientRecord = sum / base;//quotient记录进位.

tempPoint1 = tempPoint1->prev;

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = valueForBit;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

int sum = tempPoint2->data + quotientRecord;

int valueForBit = sum % base;//valueForBit为每位应当记录的的结果.

quotientRecord = sum / base;//quotient记录进位.

tempPoint1 = tempPoint1->prev;

resultTempPoint->data = valueForBit;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

②减法

subtractionAuxiliaryFunction——减法辅助函数

subtraction\_bigInteger——减法运算函数

subtraction\_bigInteger\_binarySystem——用于二进制的减法运算函数

减法的实现，是通过列竖式的思路实现的，逐位相减。

对于10进制，每位相减小于0则向前借位。

对于2进制，每位相减小于0则向前借位。

设计两次循环，第一次循环循环位数少的大整数位数次，第二循环循环差位次。与加法所不同的是减法所需要拆分的情况较多。总的来看，减法分三种情况：第一个大整数位数多于第二个大整数；第一个大整数位数小于第二个大整数；两个大整数的位数相等。然而在第三种情况中又被细分为另外三种情况：第一个长整数大于第二个长整数；第一个长整数小于第二个长整数；两个长整数大小相等。在循环过程中主要设计了两个变量用于记录循环过程中的各个数据变量。有sum记录每位相减的差以及subRecord记录借位或换句话来说退位数。设计思路大体来看符合整数减法设计思路，即为不足向前借位相减。

相关实现代码如下：

if ((tempPoint1->data - subRecord) < tempPoint2->data) {//需要借位.

//sum记录每位的差.

sum = tempPoint1->data + base - tempPoint2->data - subRecord;

subRecord = 1;//subRecord记录借位.

} else {//不需要借位.

sum = tempPoint1->data - tempPoint2->data - subRecord;//sum记录每位的差.

subRecord = 0;//subRecord记录借位.

}

tempPoint1 = tempPoint1->prev;

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = sum;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

if (tempPoint1->data < subRecord) {//若此位为0.

//从前一位借1,sum记录每位相减的差.

sum = tempPoint1->data - subRecord + base;

subRecord = 1;//sub记录借位.

} else {//若此位不为为0.

sum = tempPoint1->data - subRecord;

subRecord = 0;//sub记录借位.

}

tempPoint1 = tempPoint1->prev;

resultTempPoint->data = sum;

resultTempPoint=resultTempPoint->prev;

对于加法和减法这两种运算来说，本质两者可以看作是同一种操作，因为减法就是加上负数的加法类似操作。因此在具体实现上，基本框架是一致的（如图3所示），即首先创建一个两个操作数中较大的长度的双向链表，然后进行较小长度次循环，对三个数字进行相加或相减：两个操作数的相应位置上的数字和上一轮运算中留下的余数（需要进位的数）。然后进行相差位数的操作，保证更长的那个操作数能完成运算。

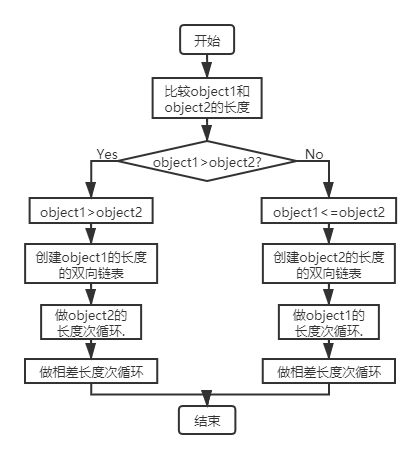


图3 线性运算（加法、减法）计算框架示意图

③乘法

multiplicationAuxiliaryFunction——乘法辅助函数

multiplication\_bigInteger——乘法运算函数

multiplication\_bigInteger\_binarySystem——用于二进制的乘法运算函数

乘法的实现采用模拟竖式运算（运算框架见图4）。在单位数字相乘后，对10或2进行取模和取余运算，得到需要进位的数字和需要保存的数字，循环进行运算。在细节处理中，乘10相当于在链表前面添加0；同时，考虑到2进制中只有0与1，所以对于每位相乘，只需简化为：

若第i位为1，则只需相加对应乘数；

若第i位为0，则只需加0。

乘法具有两种实现方法，第一种是连续的相加操作，但是这里并未采用加法的连续相加，因为那样虽然思路清晰、算法简单、实现简便，但是程序运行效率极其低下。与加减法不同的是乘法过程中运用了两重循环而并非两次循环。外层循环执行第二个因数的位数次，内层循环执行第一个因数的位数次。设计思路与整数乘法思路类似。即为从右边起依次用第二个因数的各位去乘第一个因数的各个数位，乘到哪一位，得数的末尾就和第二个因数的哪一位对齐，让后把几次相乘的结果相加。整数末尾有0的乘法直接跳出本次双重循环，因为原链表已被初始化为全0链表。

相关实现代码如下：

for (int i = 0; i < objectLength2; ++i) {

if (tempPoint2->data == 0) {

tempPoint2 = tempPoint2->prev;

tempPoint = tempPoint->prev;

markPoint = tempPoint;

continue;

}

int temp = 0;

//tempPoint1指向object1表尾节点.

node \*tempPoint1 = object1.returnHeadPoint()->prev;

for (int j = 0; j < objectLength1; ++j) {

//每次需要相加三部分数字.

temp = temp + (tempPoint2->data) \* (tempPoint1->data) + markPoint->data;

markPoint->data = temp % base;//结果.

temp = temp / base;//进位.

tempPoint1 = tempPoint1->prev;

markPoint = markPoint->prev;

}

markPoint->data = markPoint->data + temp;//进位后的结果记入链表.

tempPoint2 = tempPoint2->prev;

tempPoint = tempPoint->prev;

markPoint = tempPoint;

}

if (result.returnHeadPoint()->next->data == 0)//最大数位为0.

result.erase(1);//删除0.

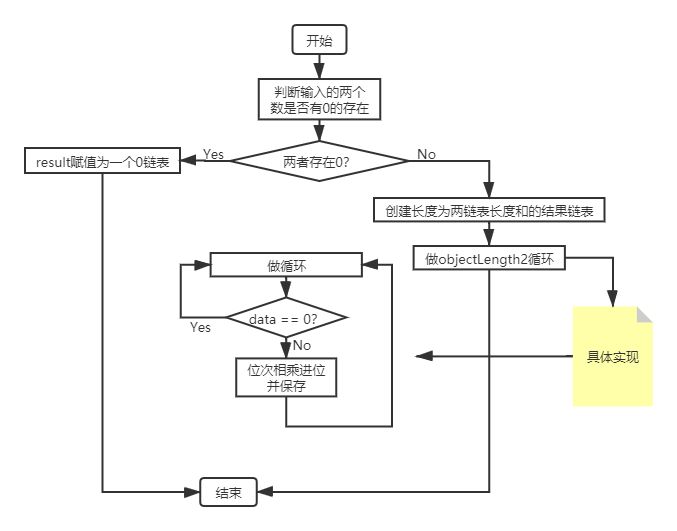


图4 乘法运算计算框架示意图

④除法

divisionFunction——除法子函数

divisionAuxiliaryFunction——除法辅助函数

division\_bigInteger——除法运算函数

division\_bigInteger\_binarySystem——用于二进制的除法运算函数

除法的实现（运算框架见图5和图6），主要思想为将除数位数与被除数位数补齐，再进行相减，随后逐次减去补位的 0，获得最终结果。

除法运算涉及到了连减运算，除数是几位，先看被除数的前几位，前几位不够除，则多看一位，除到哪位，商就写在哪位上面，不够则商1，0用于占位。余数要比除数小。在本程序的运行中，设置了双重循环。外层循环的循环次数正好为商的位数，而内层循环则负责连续做减法。在循环过程中，设计了两个变量来记录循环过程中的各个数据。链表对象tempList,将每次减法后得到的差赋给tempList，tempList再作为被减数继续做连减操作。整型变量count用于记录执行连减运算的次数，每一轮外层循环后得到的次数的值即为商的每一位，而最后得到的链表tempList即为余数。 故在计算商的过程中自然而然就将余数也一并计算得到了。

相关实现代码如下：

if ((objectLength2 - 1) != 0) {

tempList.createLinkListWithoutForValue(objectLength2 - 1);

node \*tempListPtr = tempList.returnHeadPoint()->next;

node \*ptr = object1.returnHeadPoint()->next;

for (int i = 0; i < objectLength2 - 1; ++i) {//循环赋值.

tempListPtr->data = ptr->data;

tempListPtr = tempListPtr->next;

ptr = ptr->next;

}

}

node \*markPtr = object1.returnHeadPoint()->next;

//寻找object2最高位位于object的位置.

for (int i = 0; i < objectLength2 - 1; ++i) {

markPtr = markPtr->next;

}

node \*erasePoint;

for (int i = objectLength2 - 1; i < objectLength1; ++i) {

int count = 0;//count记录相减次数.

erasePoint = tempList.returnHeadPoint()->next;

if (erasePoint->data == 0) {//相减之后差为0,将0后的数代替0.

node \*change = tempList.returnHeadPoint()->next;

change->data = markPtr->data;

} else {//继续插入下一位数.

tempList.insert(markPtr->data, tempList.linkListSize());

}

while ((comparisonOfSize(tempList, object2)) != -1) {//比较两个数的大小.

tempList = divisionFunction(tempList, object2, base);//做连减操作.

count++;//记录连减次数.

}

markPtr = markPtr->next;

//将每次连减的次数插入到result中.

result.insert(count, result.linkListSize());

}

if (result.returnHeadPoint()->next->data == 0) {//若最大数为0.

result.erase(1);

}

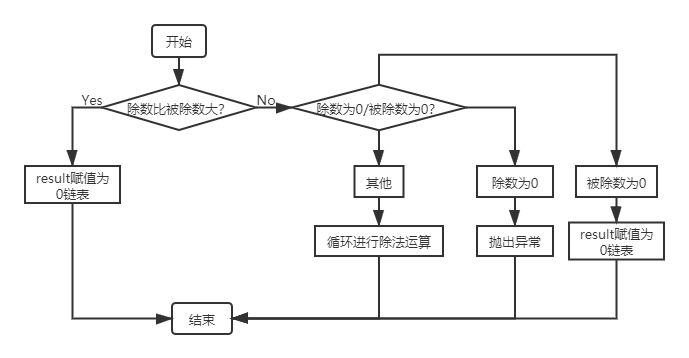


图5 除法运算计算框架示意图

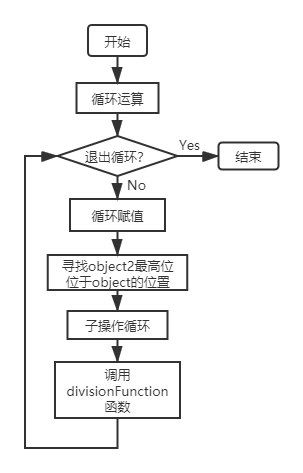


图6 循环除法运算子框架

⑤指数运算

指数运算通过调用乘法辅助函数进行循环连乘，通过除法子函数控制循环次数的更新（运算框架见图8）。

相关实现代码如下：

tempList.insert(1, 0);

result = multiplicationAuxiliaryFunction(operation1, operation1, base);//做一次乘法,平方.

operation2 = divisionFunction(operation2, tempList, base);//循环次数减1.

while (comparisonOfSize(operation2, tempList) == 1) {

result = multiplicationAuxiliaryFunction(result, operation1, base);//连乘.

operation2 = divisionFunction(operation2, tempList, base);//循环次数减1.

}

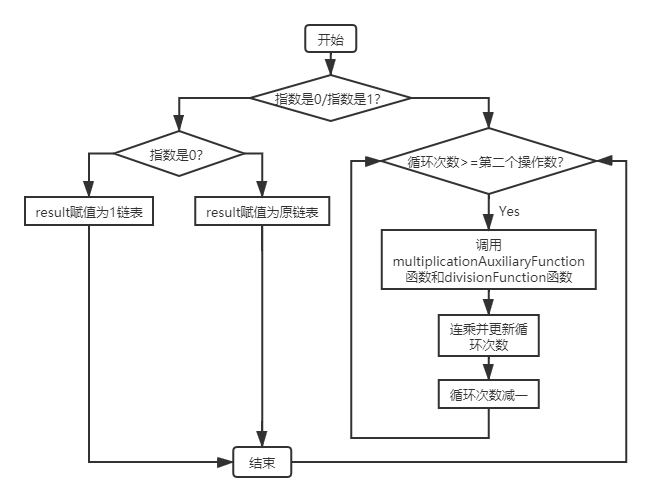


图7 指数运算计算框架示意图

表8 action类

|  |  |
| --- | --- |
| action类 | |
| 功能 | 用于系统数据的输入输出运算操作 |
| 静态成员函数 | |
| static void welcomeScreen()  欢迎界面 | |
| static void directorySelectionScreen()  目录选择界面 | |
| static void baseSelection()  进制选择 | |
| static void exitSystem()  退出程序 | |
| static void selectionDirectory()  运算方法 | |
| static void operationWithoutFile()  不涉及文件的大整数计算 | |
| static void operationWithFile()  涉及文件的大整数计算 | |

welcomeScreen函数用于展示欢迎界面，主要是展示一些个人信息和系统操作方法；directorySelectionScreen函数用于展示是否涉及文件进行操作的选择目录；baseSelection函数展示进制选择目录，选择十进制操作或者二进制操作；exitSystem函数用于退出整个系统；selectionDirectory函数用于展示运算操作类型选择目录。

operationWithoutFile函数是不涉及文件的大整数计；operationWithFile函数是涉及文件的大整数计算。两者的计算流程类似（实现流程见图8）。

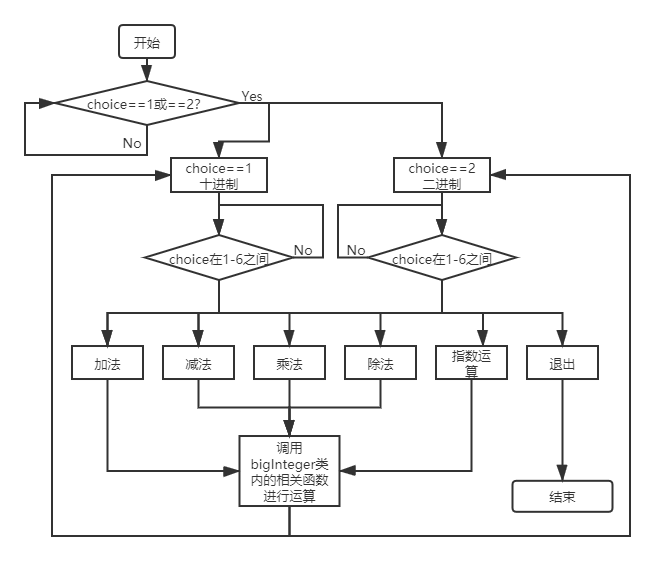


图8 选择操作流程

## （4）主程序的设计

主程序通过对类的方法调用实现不同的功能。action类内设置为静态方法，因此不需要创建对象，直接使用类名进行调用（操作流程见图9）。

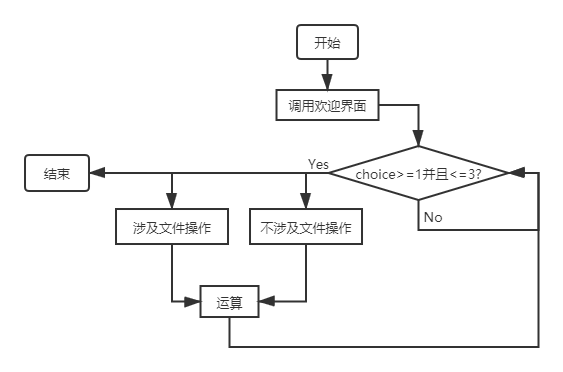


图9 主程序操作流程

# 四、调试分析

## （1）技术难点分析

（1）问题描述：采用何种数据结构来存储并实现大整数的各种运算。

解决方法：采用带有头节点的双向循环链表存储数据，并实现大整数运算。链表的每一个节点中存放一个整型数据，无论是插入删除还是倒序遍历链表，双向循环链表都比较方便。

（2）问题描述：在类内定义输出函数重载输出流出现错误。

解决方法：在类外重载输出函数，将display(ostream &out)函数定义为链链表类内的成员函数，并用重载的输出函数调用display()函数实现链表中数据的直接输出。

（3）问题描述：结果输出文件时，文件位置的多样化。

解决方法：将文件路径url设置为一个string类型字符串，并作为函数参数传进即可。比如string url = R"(C:\Users\15590\Desktop\Operation of large integers\file\result\_binarySystem.txt)"

（4）问题描述：被减数小于减数，符号的输出。

解决方法：在减法运算过程中的较小数减较大数的过程中，结果会出现负值。自定义的双向循环链表的每一个节点保存一个整型变量，故”-”号无法作为整型数据插入到链表中，我们只能简单的凭借输出及逆行判断，将负号直接输出

## （2）调试错误分析

（1）在测试时，在选择涉及文件的操作分支时，不能进入大整数输入环节，一直在循环输出目录。

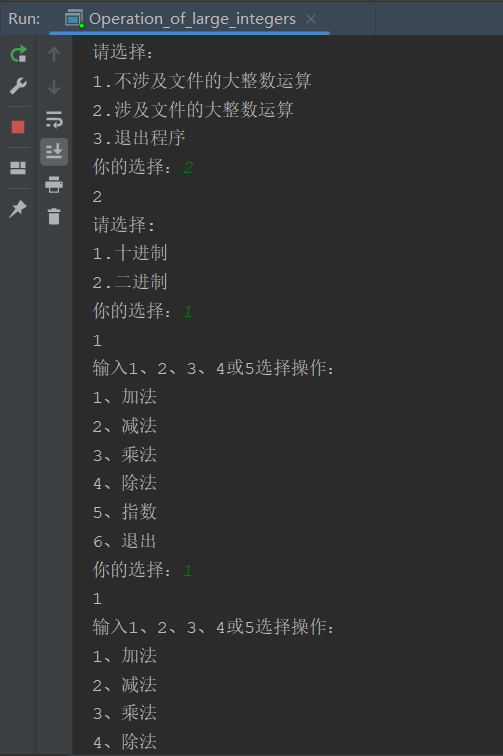


图10 调试错误（1）

解决方案：

1、分析可能为listOperateWithFile\_decimalSystem函数可能存在问题；

2、经设置断点逐步运行发现因为文件打开失败跳不出循环；

3、修改文件流打开代码。

（2）头文件异常

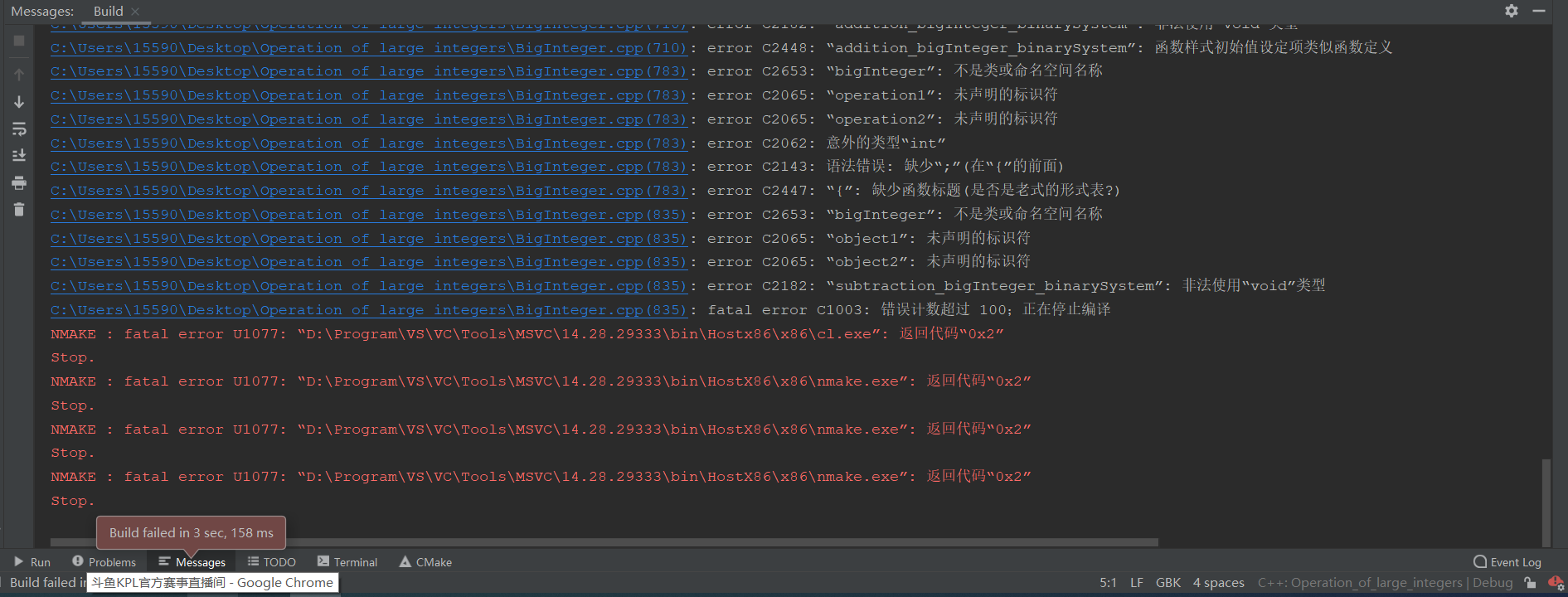


图11 调试错误（2）

解决方案：

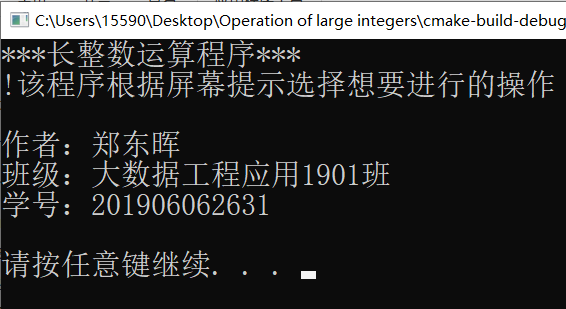
1、分析可能为变量引入方面可能存在问题；

2、检查代码文件发现因为未引入头文件；

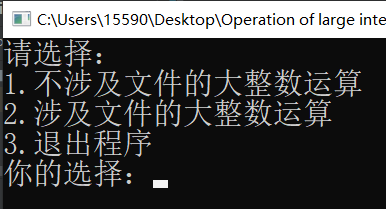
3、修改代码文件，加入正确的头文件。

# 五、测试结果分析

系统进入界面



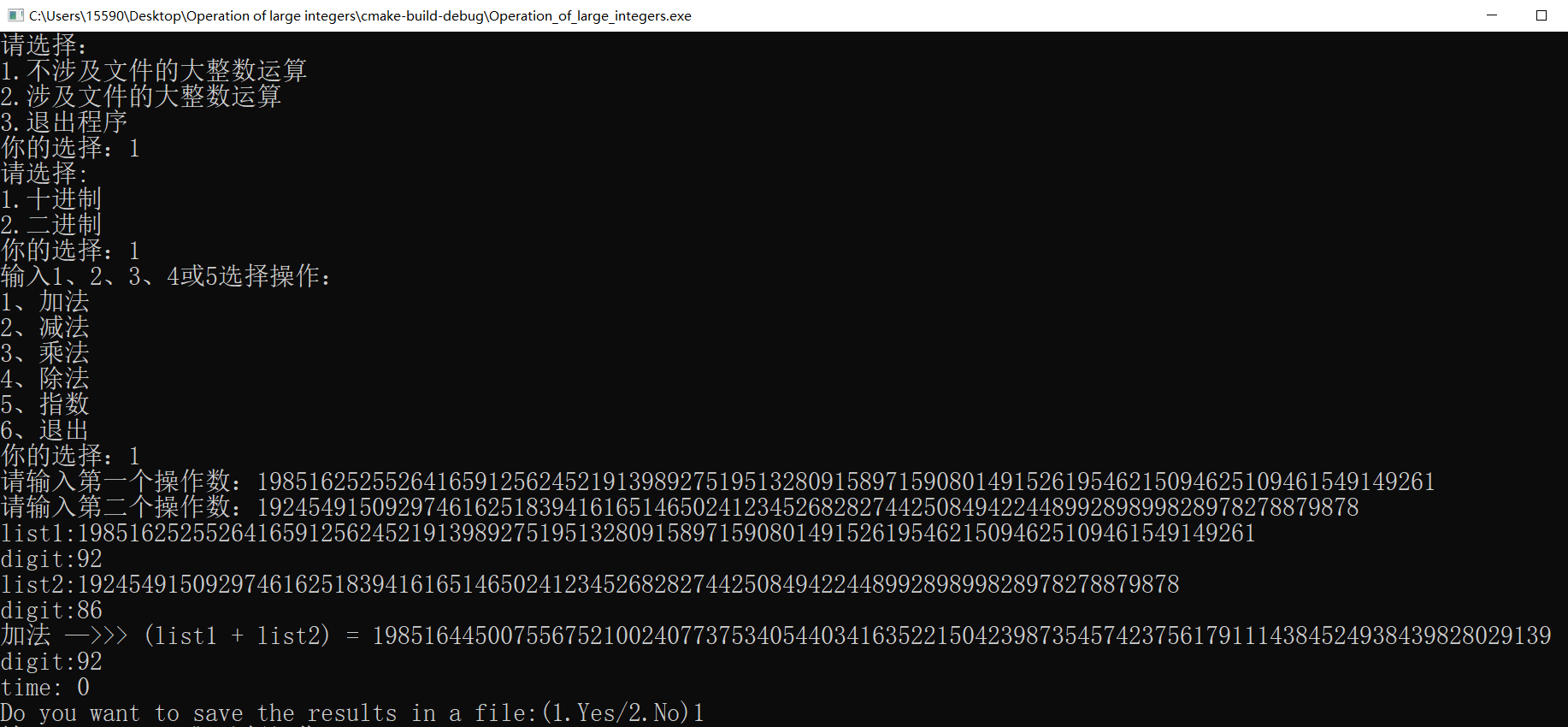
操作方式选择



## 不涉及文件输入操作

## 十进制

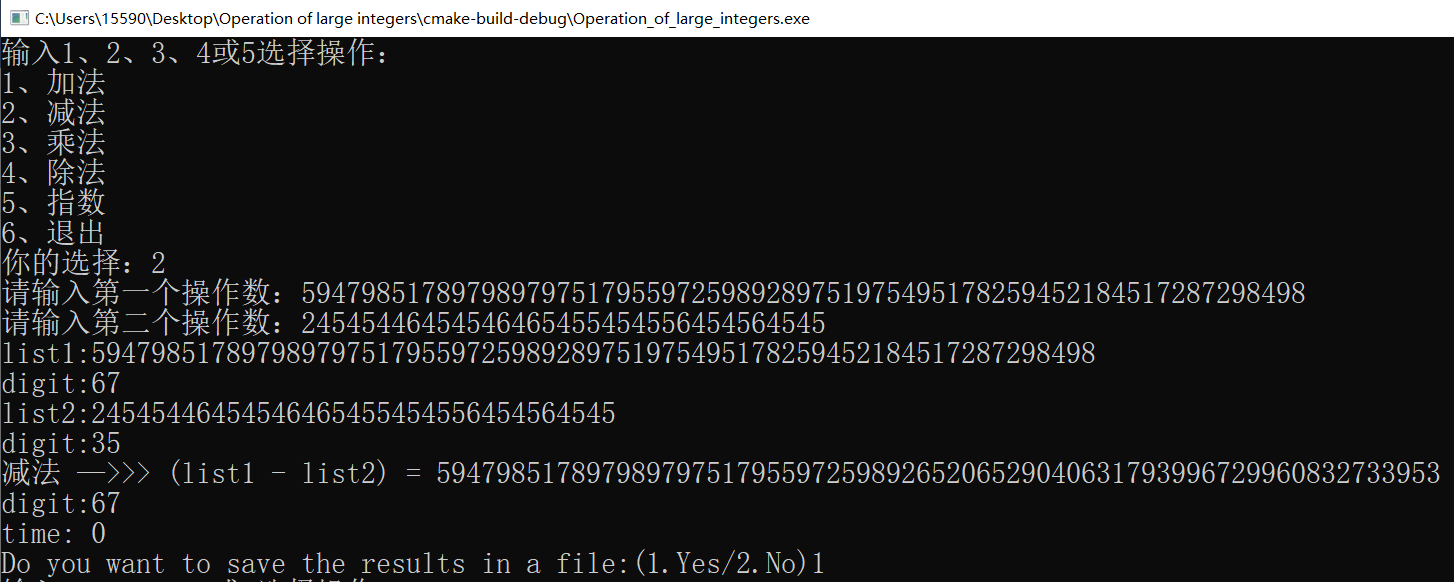
加法



经验证可以存进结果文件



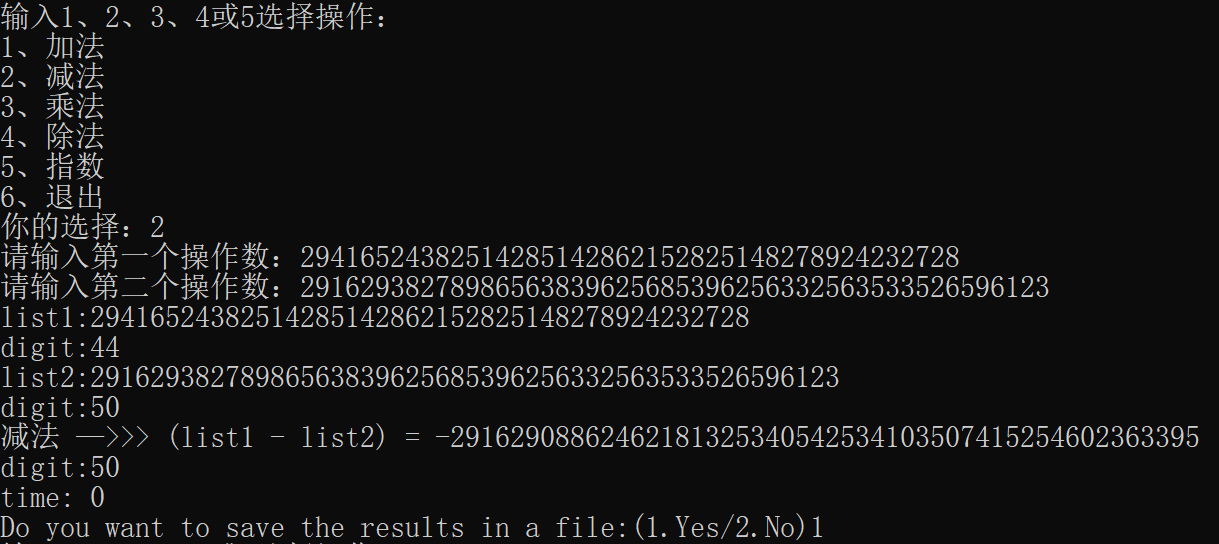
减法



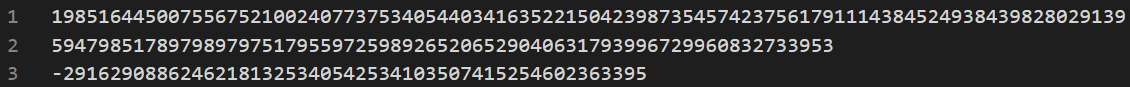
经验证可以存进结果文件



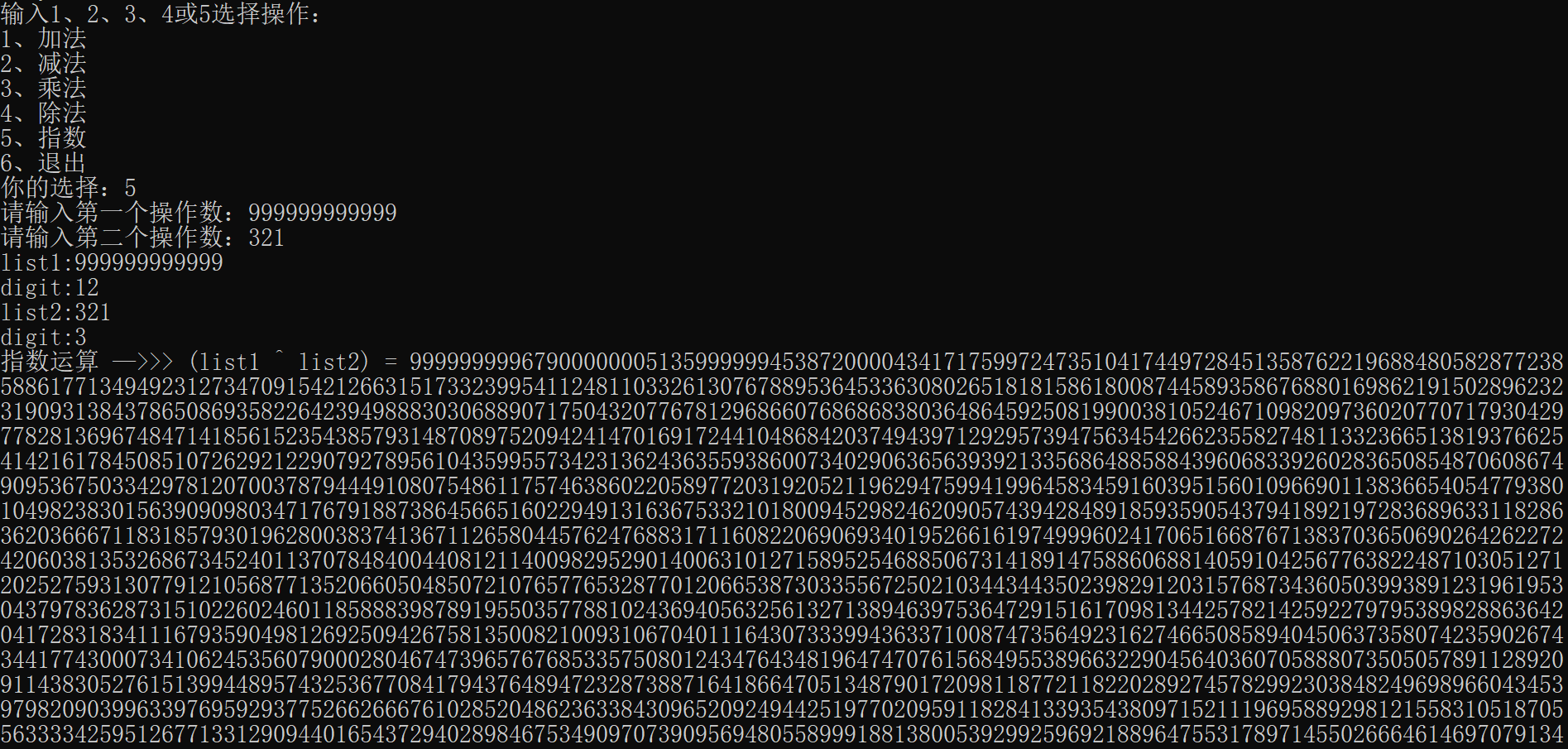
附：若被减数小于减数

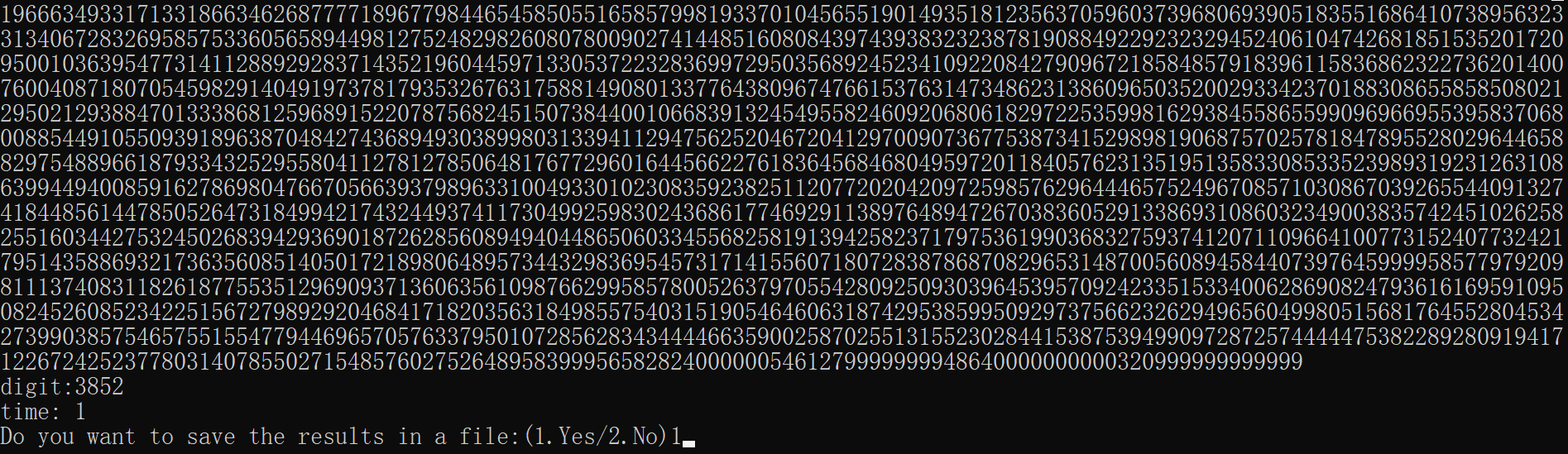


经验证可以存进结果文件



指数运算

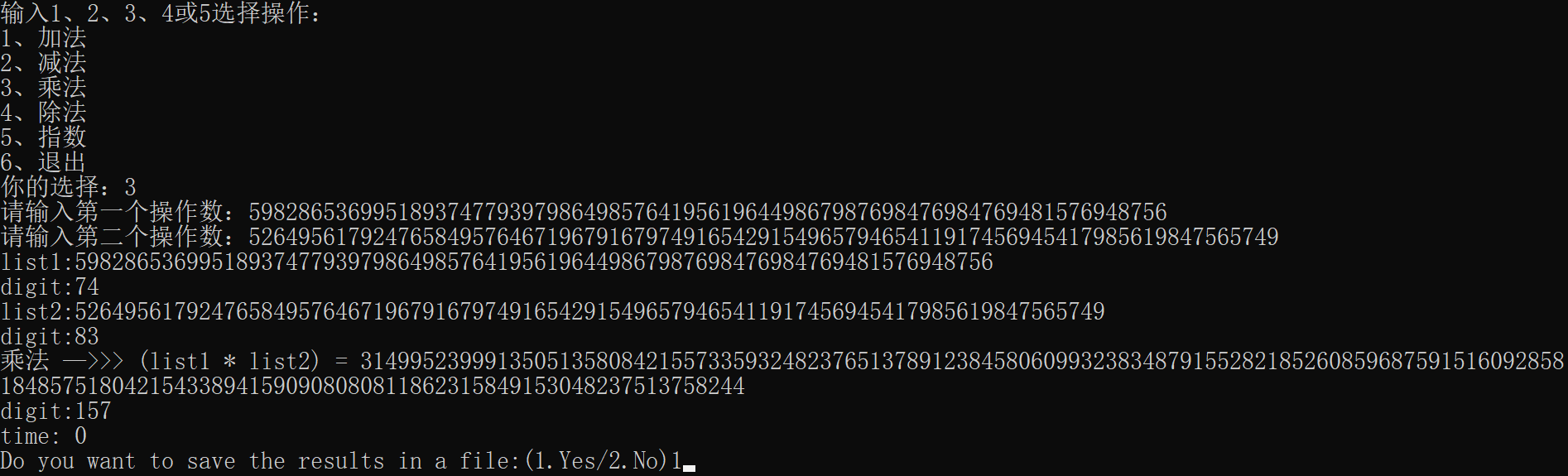




经验证可以存进结果文件



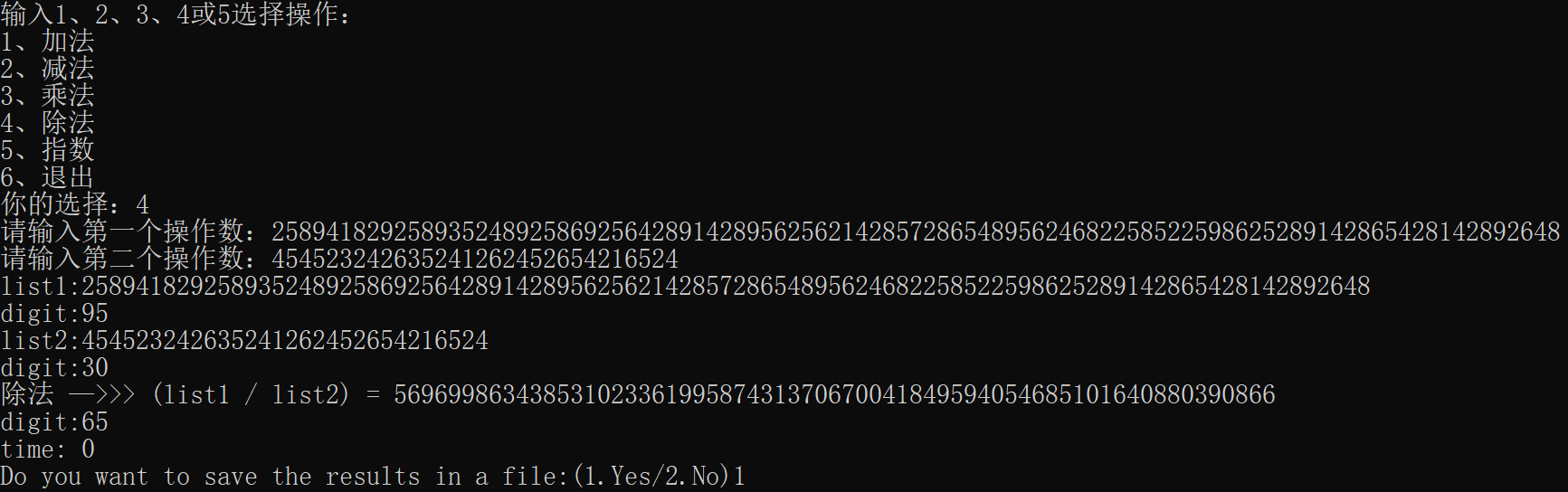
乘法



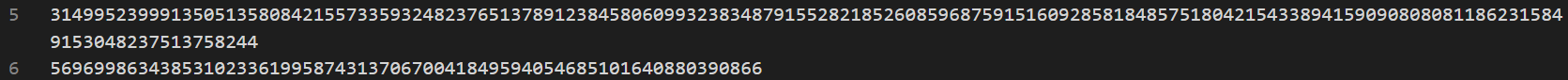
经验证可以存进结果文件



除法

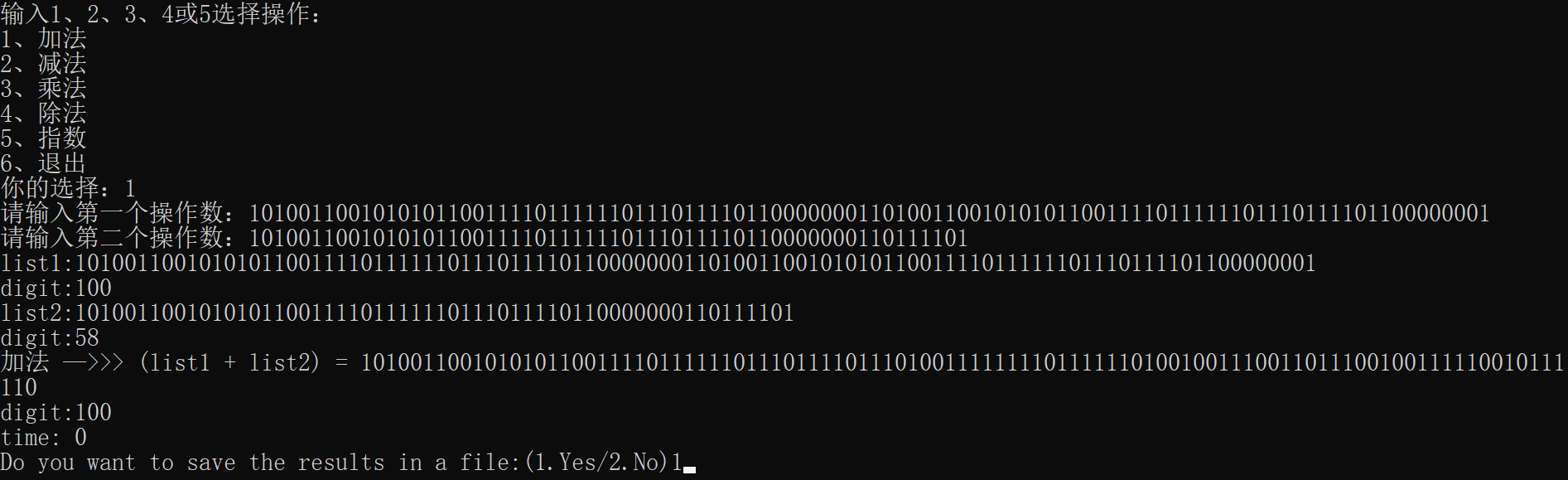


经验证可以存进结果文件



## 二进制

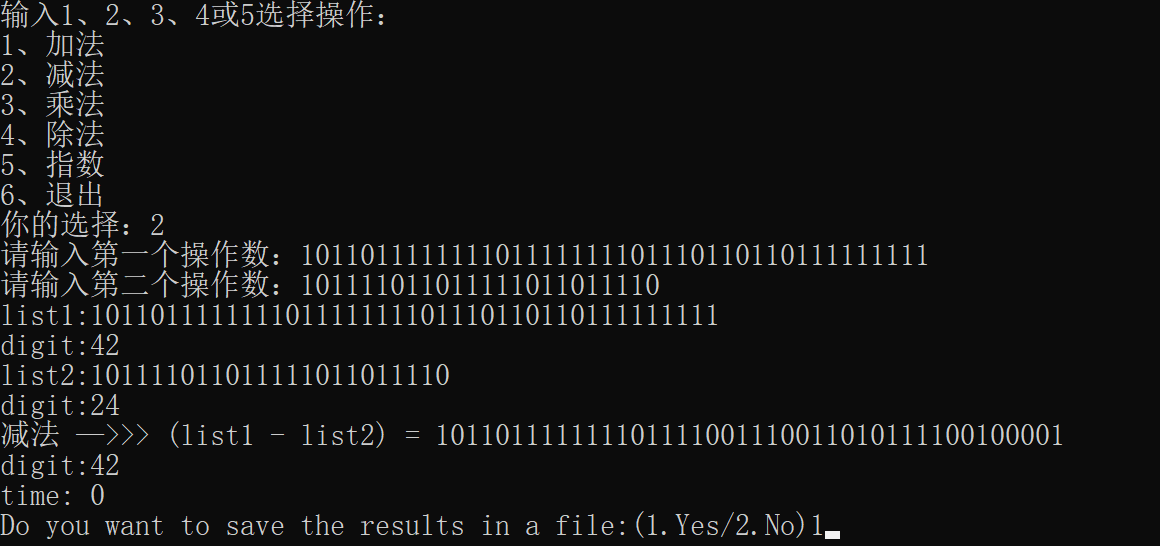
加法



经验证可以存进结果文件



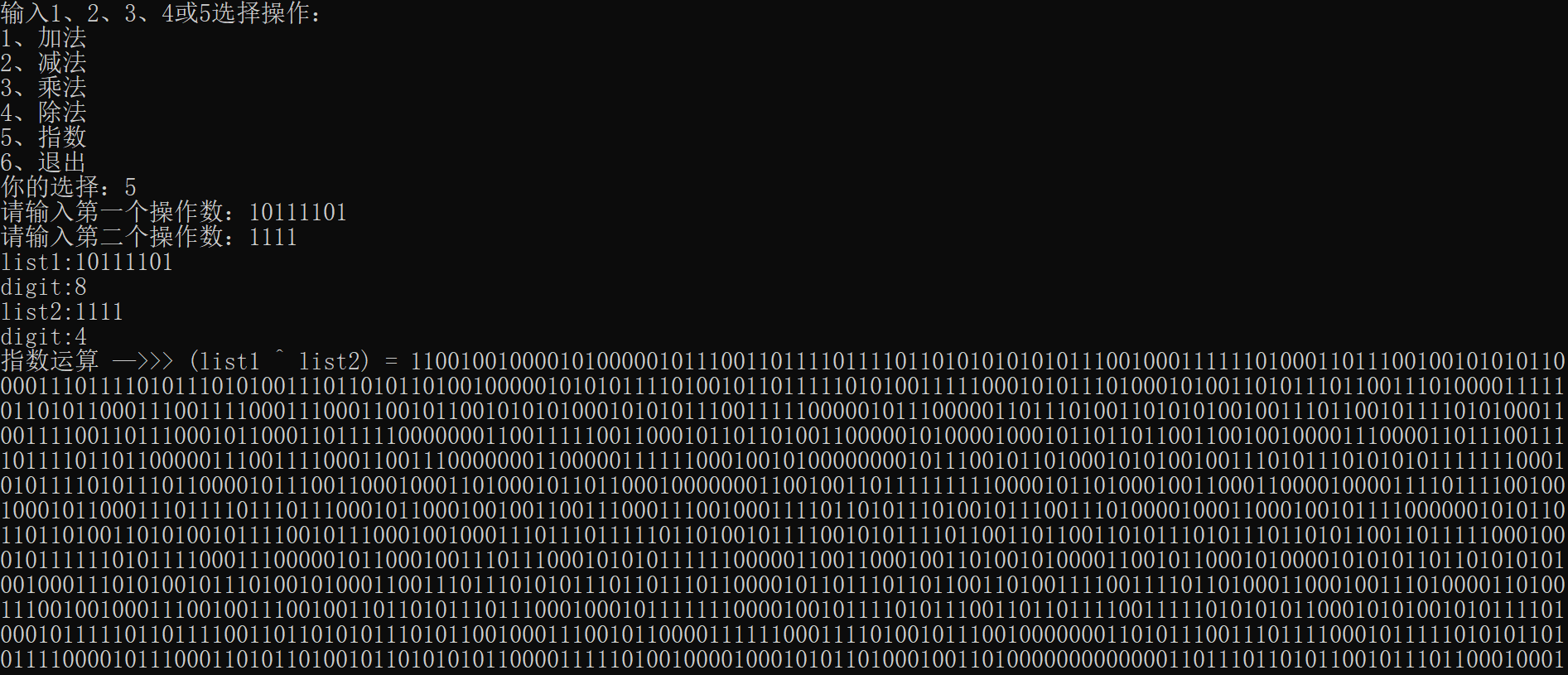
减法



经验证可以存进结果文件



指数运算

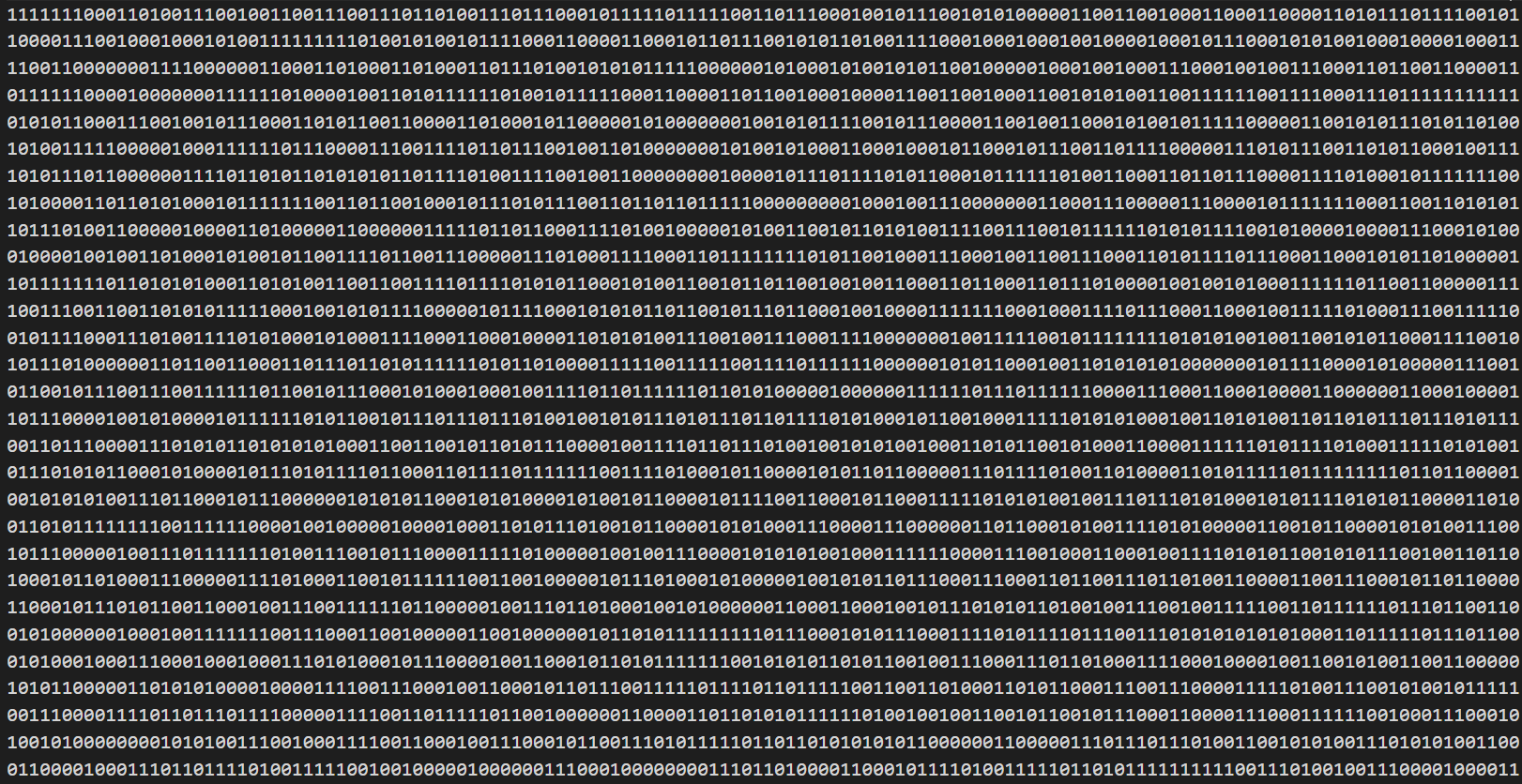


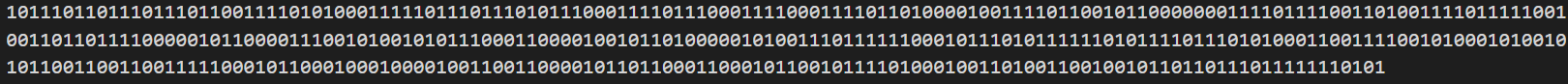




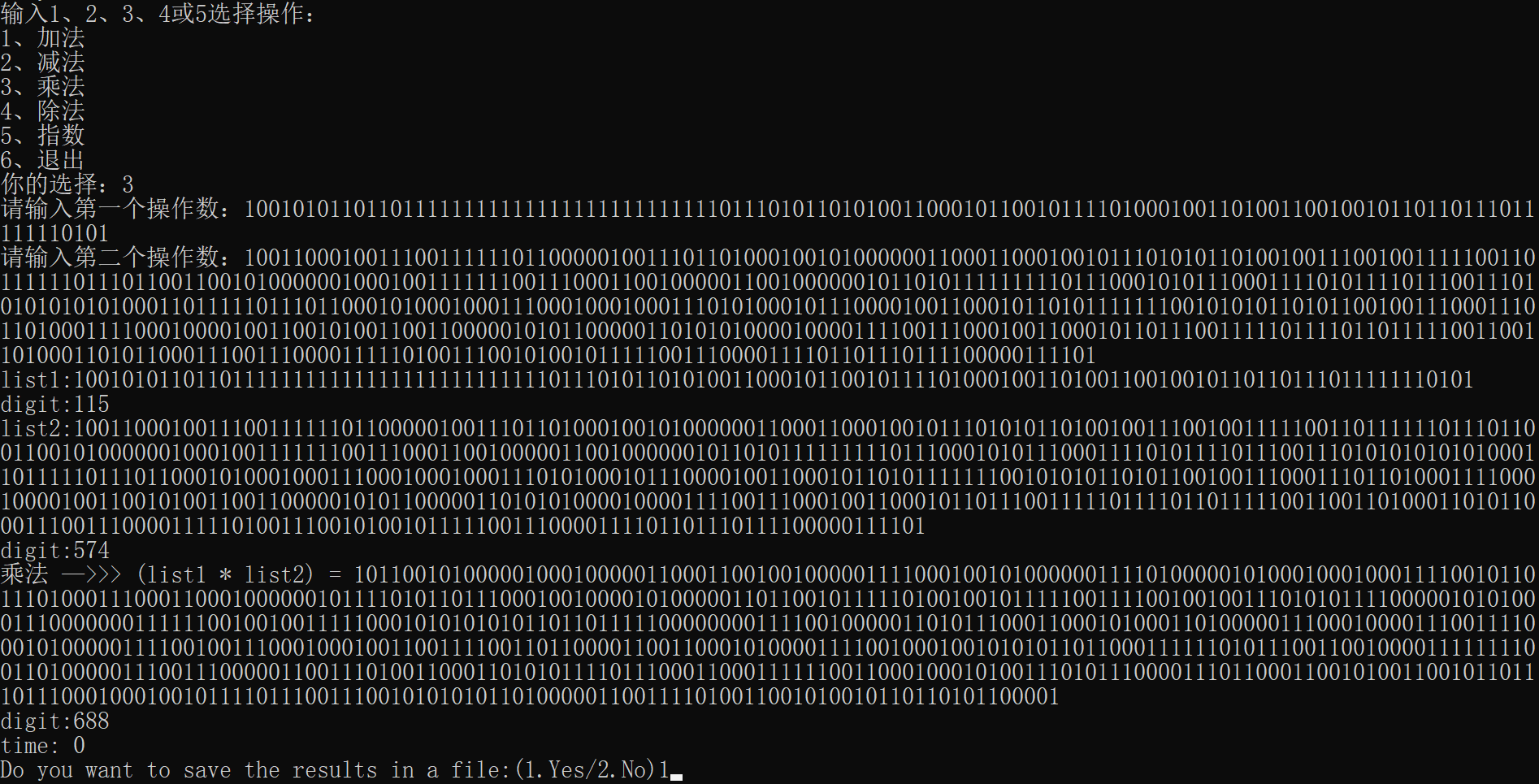
经验证可以存进结果文件



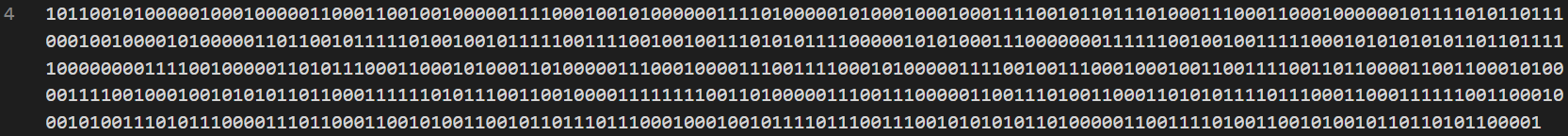




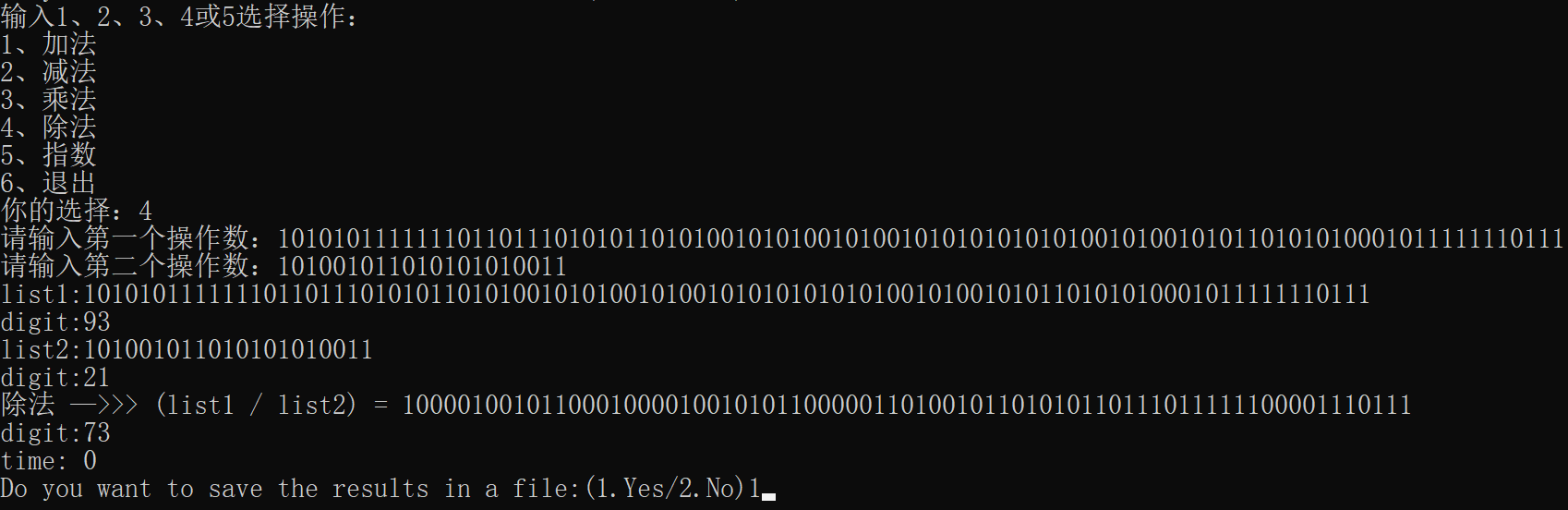
乘法



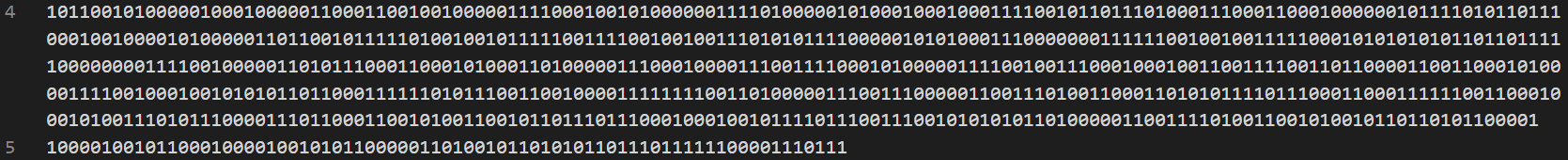
经验证可以存进结果文件



除法



经验证可以存进结果文件



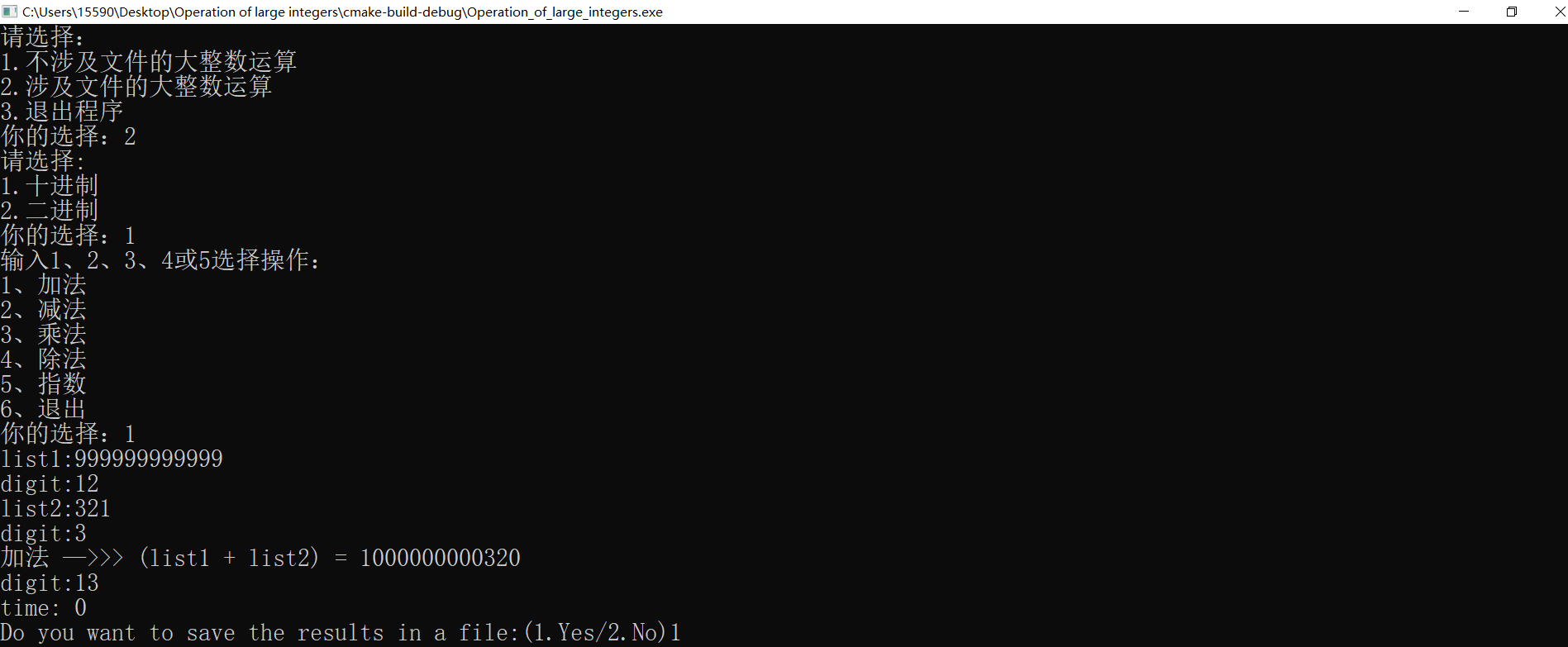
## （2）涉及文件输入操作

## ①十进制

下图为file\_decimalSystem.txt输入文件的数据，两个数据使用空格隔开，分别代表操作数1和操作数2。第一行的1是一个标记字符，1代表该文件有操作数存在，0表示该文件没有操作数存在。



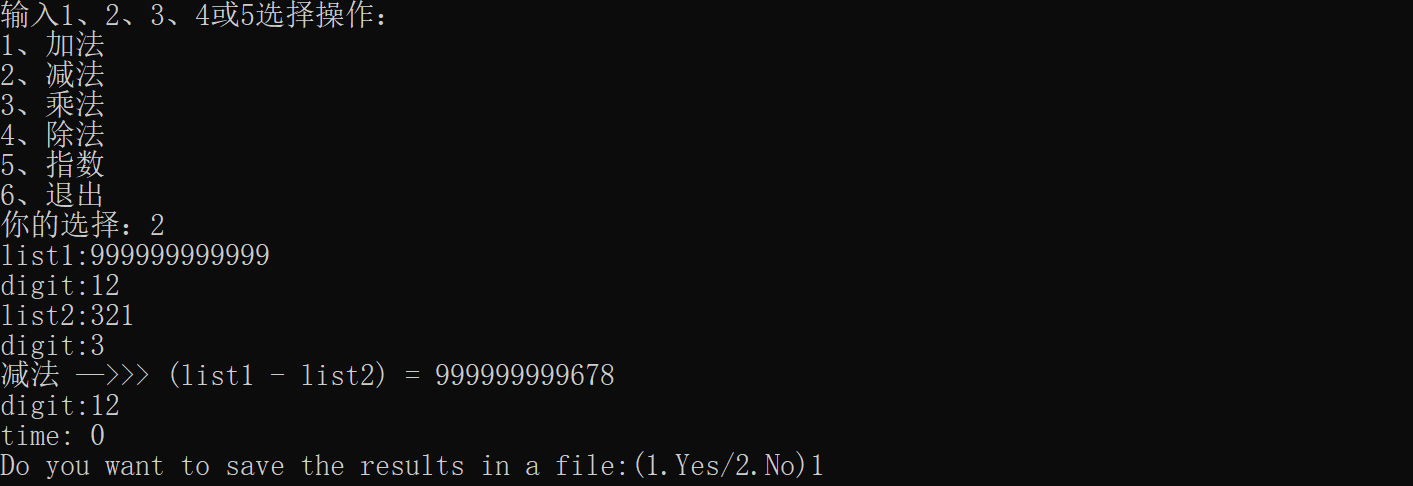
加法



经验证可以存进结果文件



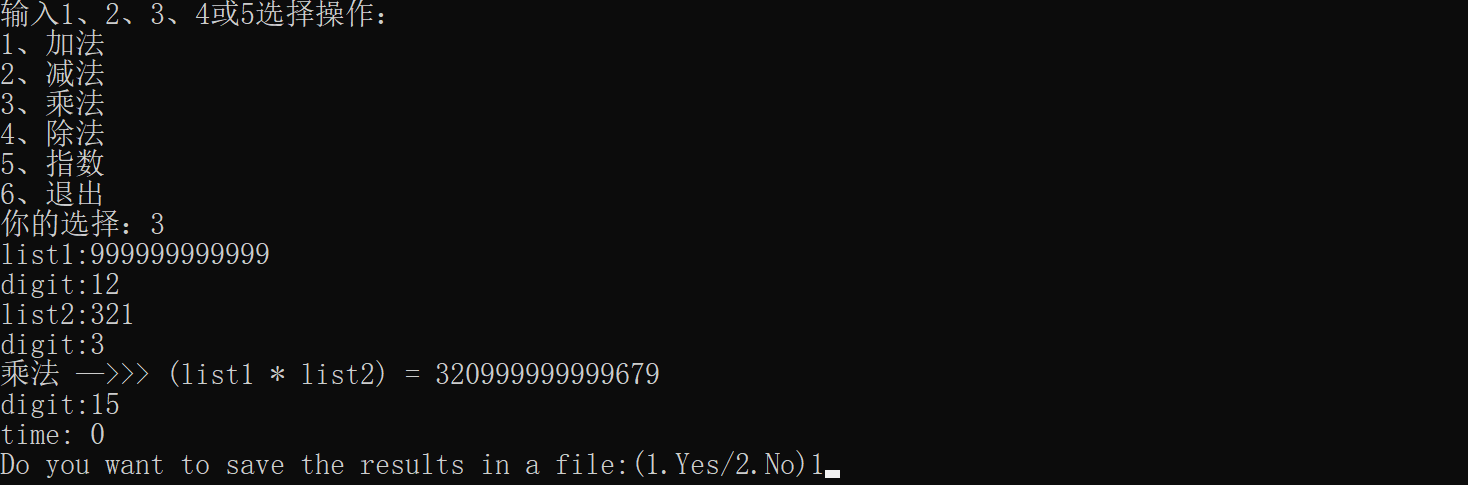
减法



经验证可以存进结果文件



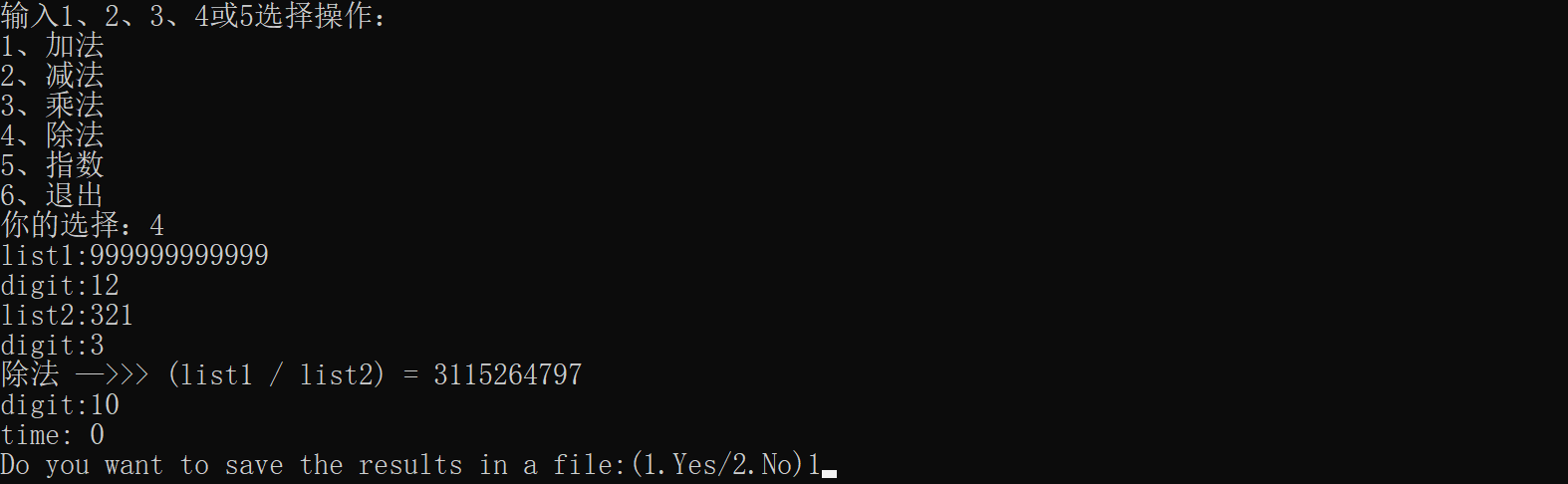
乘法



经验证可以存进结果文件



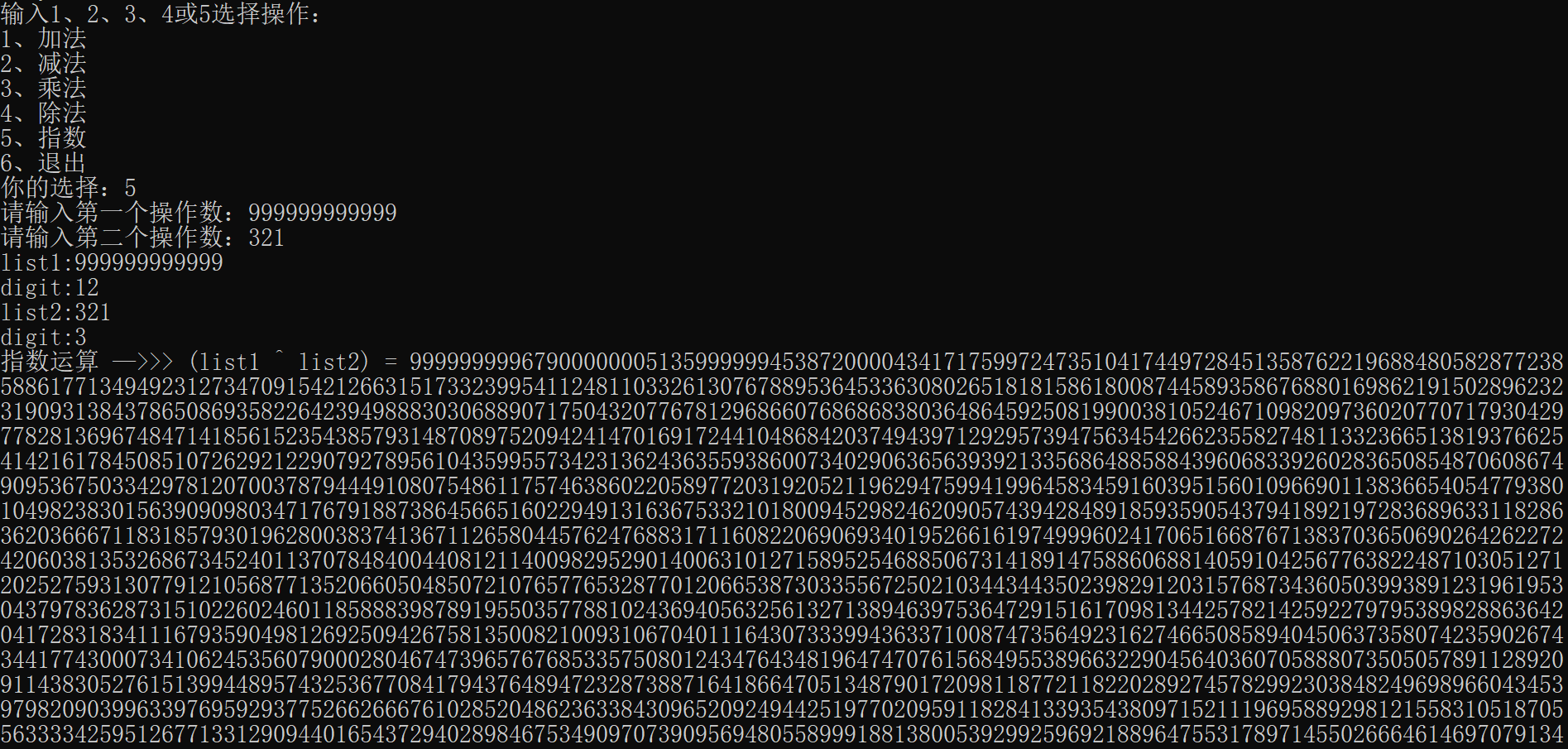
除法

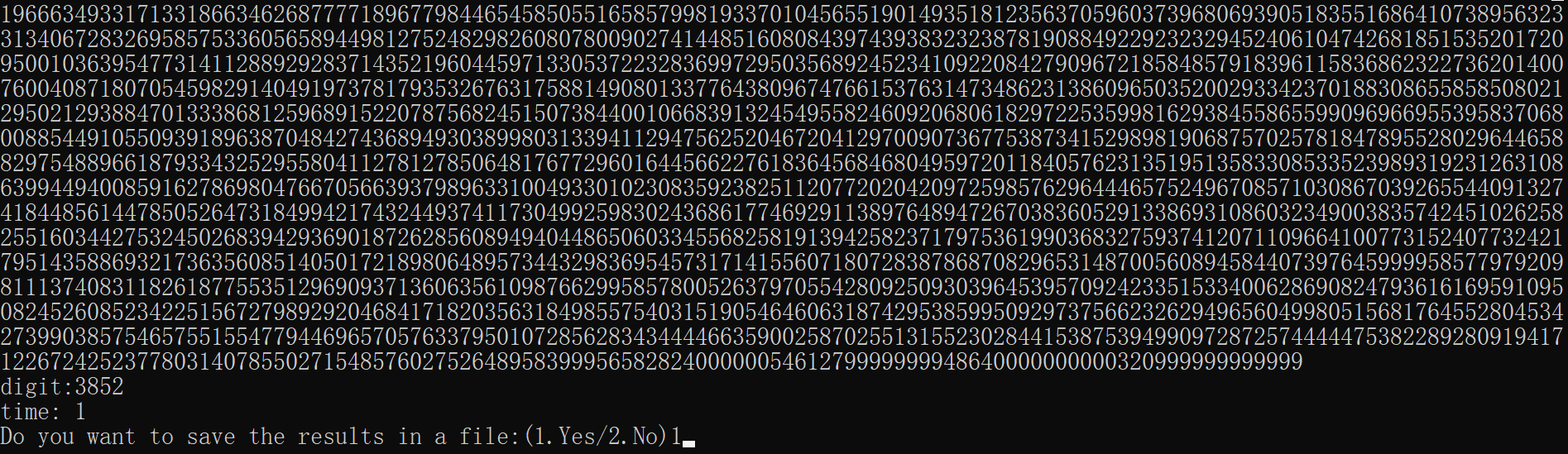


经验证可以存进结果文件

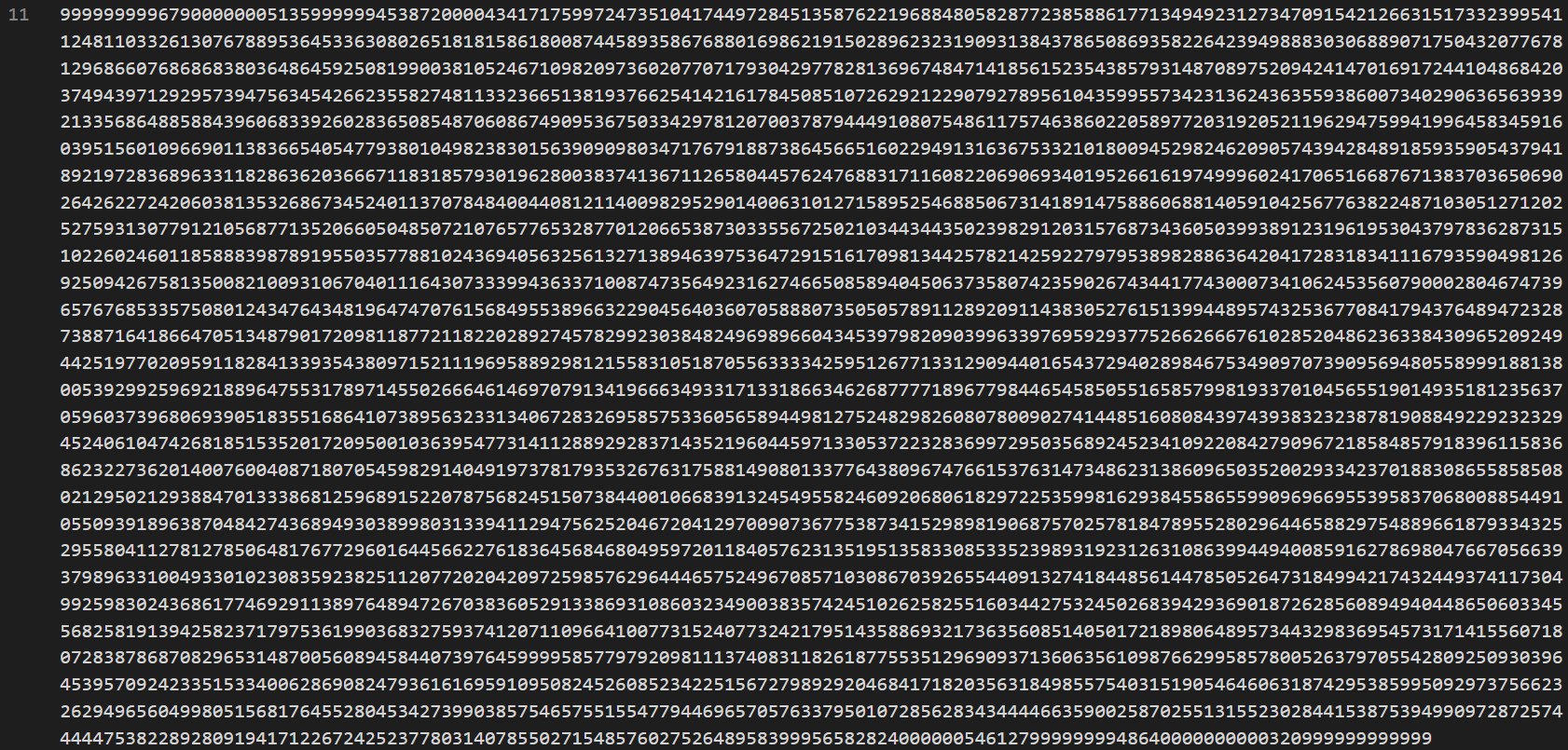


指数运算





经验证可以存进结果文件

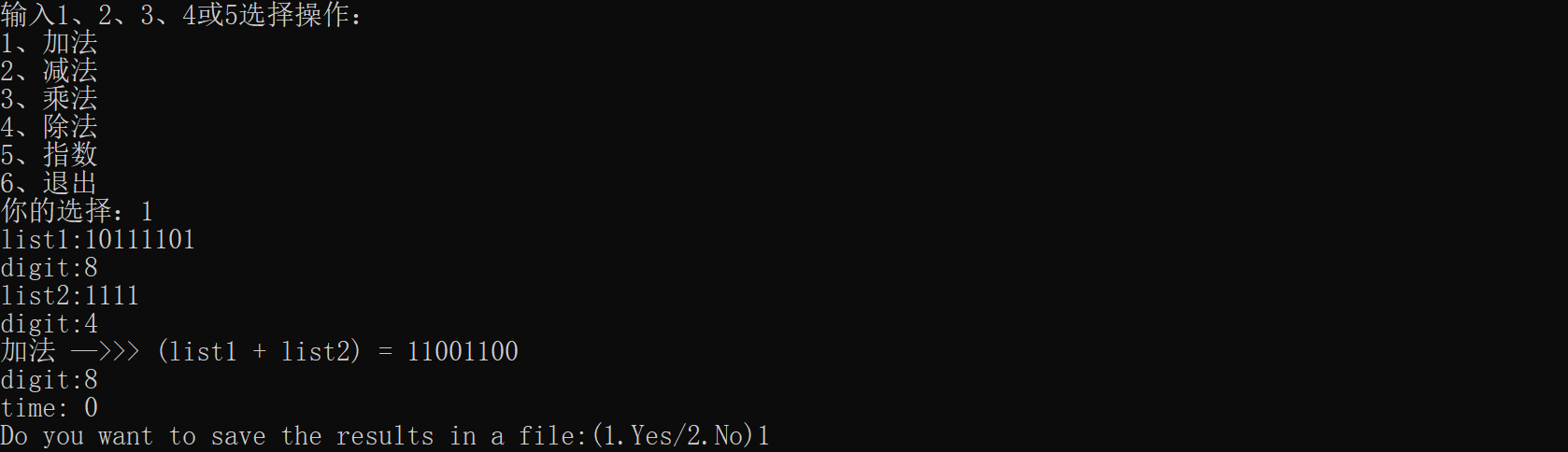


## ②二进制

下图为file\_binarySystem.txt输入文件的数据，两个数据使用空格隔开，分别代表操作数1和操作数2。第一行的1是一个标记字符，1代表该文件有操作数存在，0表示该文件没有操作数存在。



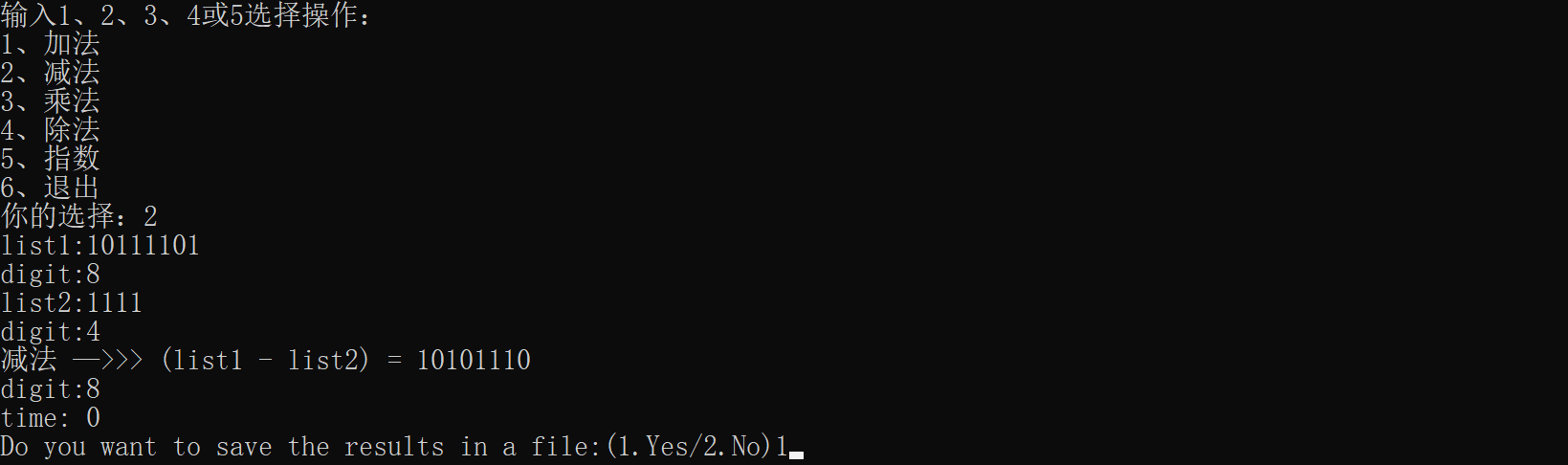
加法



经验证可以存进结果文件



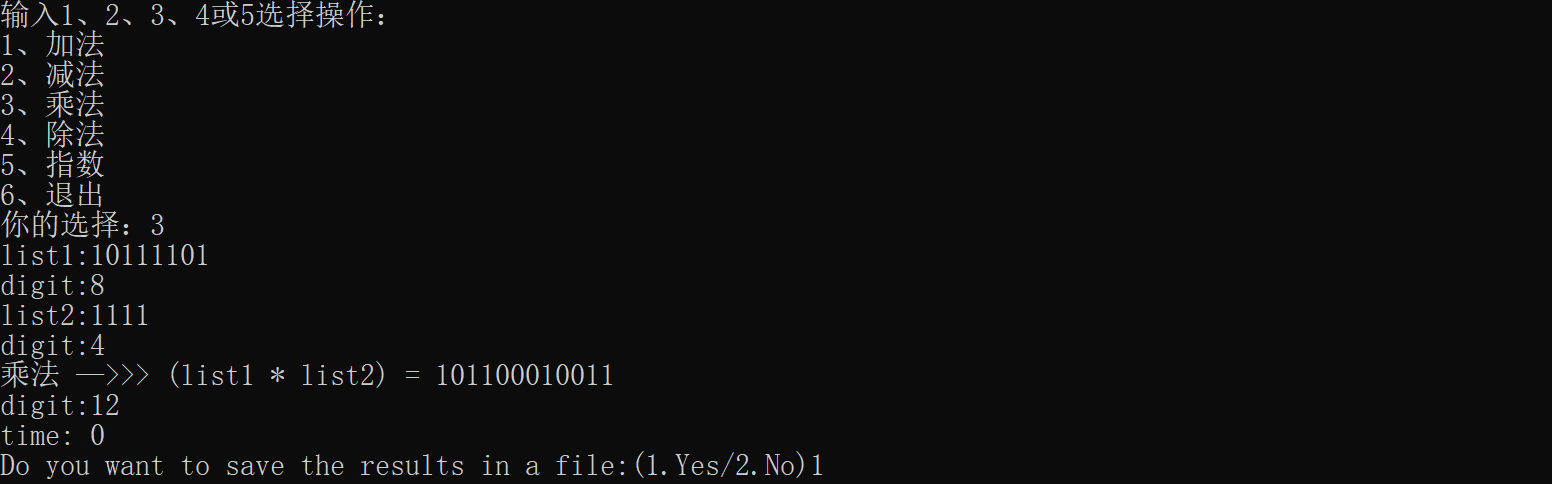
减法



经验证可以存进结果文件



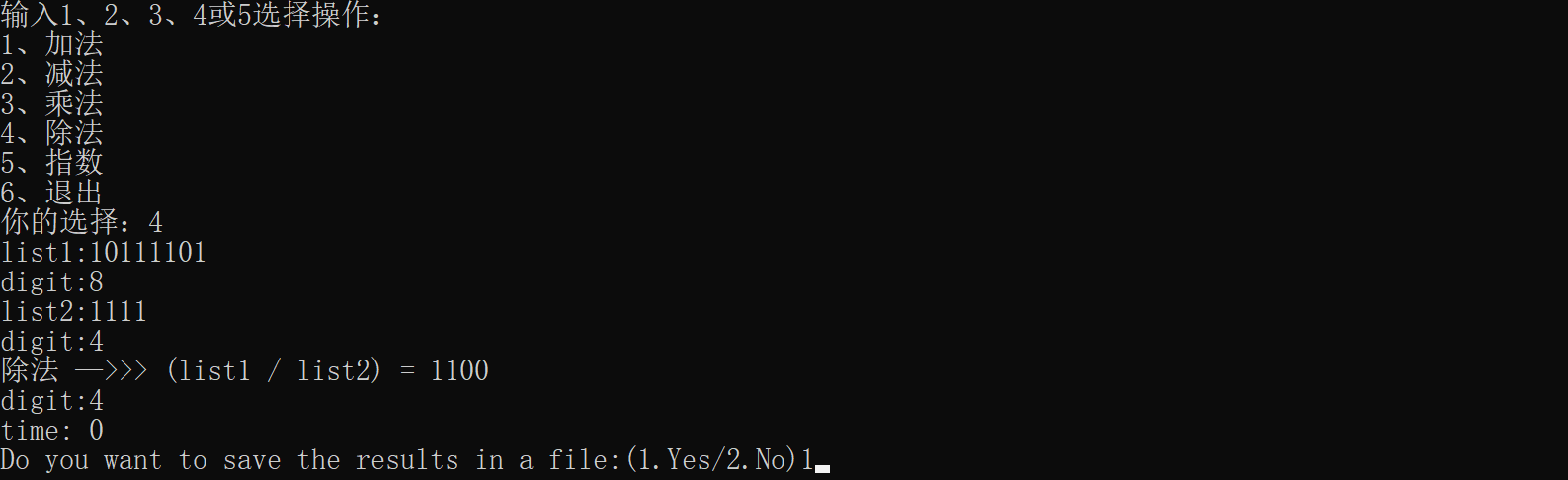
乘法



经验证可以存进结果文件



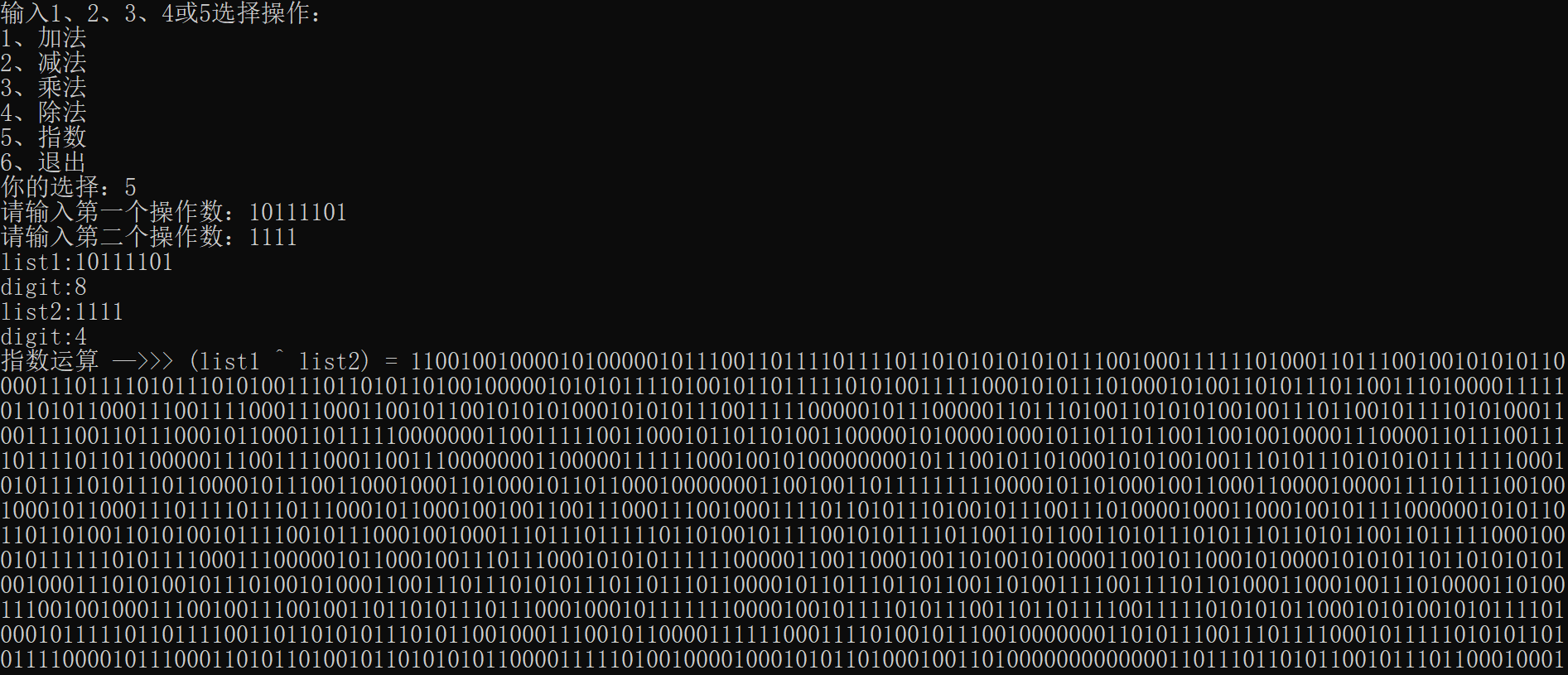
除法



经验证可以存进结果文件



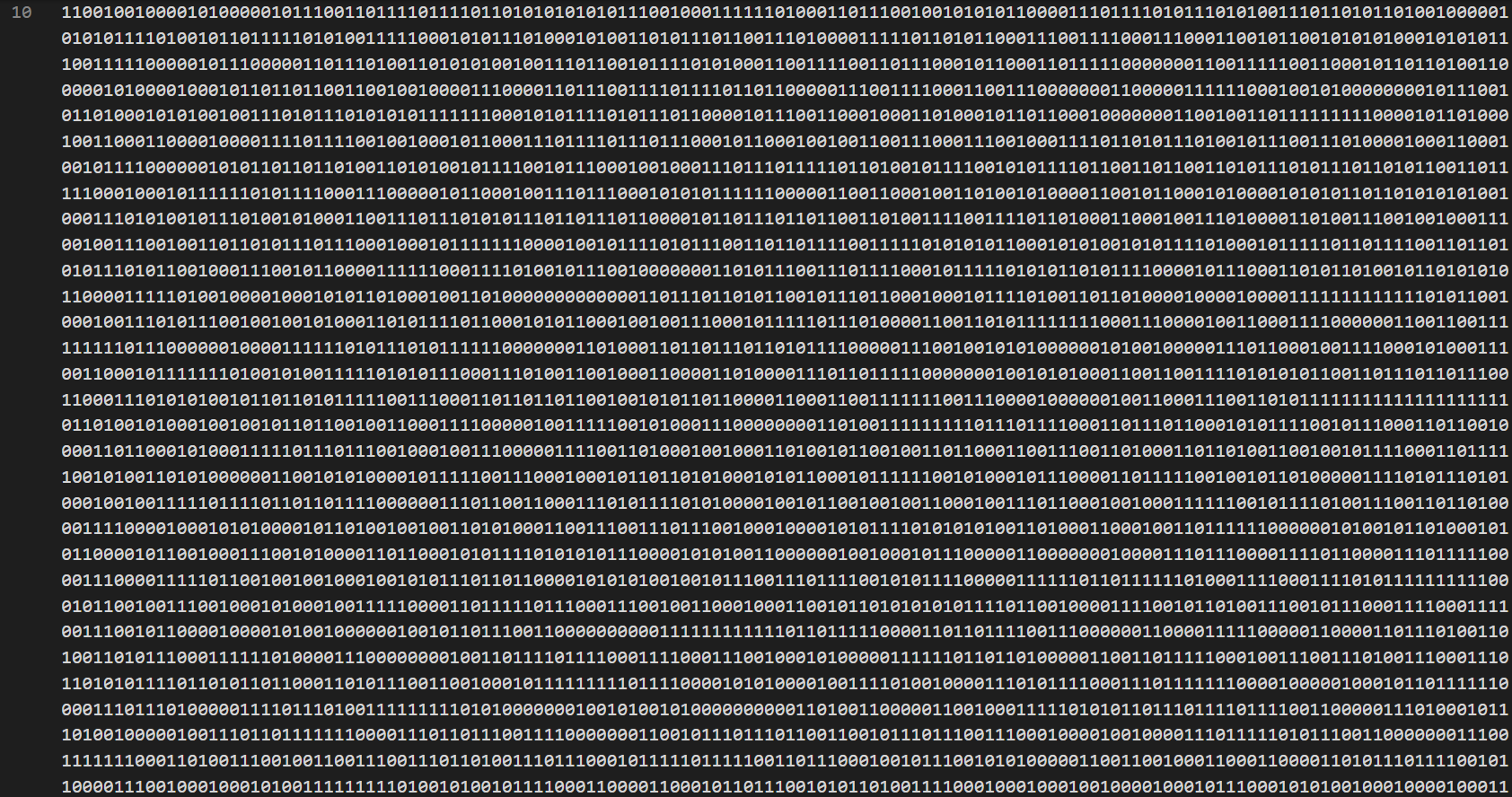
指数运算





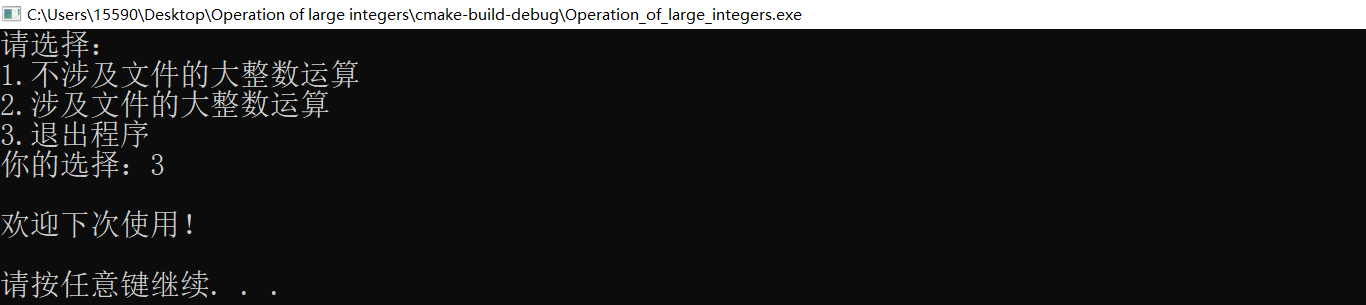


经验证可以存进结果文件





## （3）退出程序操作



# 六、附录

===============================================================================

Node.h

#ifndef OPERATION\_OF\_LARGE\_INTEGERS\_NODE\_H

#define OPERATION\_OF\_LARGE\_INTEGERS\_NODE\_H

#include <iostream>

using namespace std;

//节点类.

class node {

public:

//存储的数据.

int data;

//前驱指针和后继指针.

node \*prev, \*next;

/\*\*

\* @details 无参构造函数.

\* @param void.

\* @return

\*/

node() : data(0), prev(nullptr), next(nullptr) {}

/\*\*

\* @details 有参构造函数.

\* @param item:数据.

\* @param prevNode:前驱指针.

\* @param nextNode:后继指针.

\* @return

\*/

explicit node(const int &item, node \*prevNode = nullptr,

node \*nextNode = nullptr) :

data(item), prev(prevNode), next(nextNode) {}

};

#endif //OPERATION\_OF\_LARGE\_INTEGERS\_NODE\_H

===============================================================================

LinkList.h

#ifndef OPERATION\_OF\_LARGE\_INTEGERS\_LINKLIST\_H

#define OPERATION\_OF\_LARGE\_INTEGERS\_LINKLIST\_H

#include <iostream>

#include "Node.h"

using namespace std;

//双向链表类.

class linkList {

public:

/\*\*

\* @details 返回头指针.

\* @param void.

\* @return node\*:头指针对象.

\*/

node\* returnHeadPoint();

/\*\*

\* @details 返回链表长度.

\* @param void.

\* @return int&:返回一个引用.

\*/

int& returnLinkListLength();

/\*\*

\* @details 返回链表符号标记.

\* @param void.

\* @return int&:返回一个引用.

\*/

int& returnLinkListFlag();

/\*\*

\* @details 获取链表的长度.

\* @param void.

\* @return int:返回双向链表的长度.

\*/

int linkListSize() const;

/\*\*

\* @details 获取链表的符号.

\* @param void.

\* @return int:返回双向链表的符号.

\*/

int linkListFlag() const;

/\*\*

\* @details 无参构造函数.

\* @param void.

\* @return

\*/

linkList();

/\*\*

\* @details 拷贝构造函数,防止浅拷贝.

\* @param object:被拷贝链表对象.

\* @return

\*/

linkList(const linkList &object);

/\*\*

\* @details 赋值运算符重载.

\* @param rightHandObject:赋值运算符右对象.

\* @return linkList &:复制后的新对象.

\*/

linkList & operator=(const linkList &rightHandObject);

/\*\*

\* @details 析构函数,释放内存.

\* @param void.

\* @return

\*/

~linkList();

/\*\*

\* @details 输出函数,用于输出链表.

\* @param out:输出对象.

\* @return void.

\*/

void display(ostream &out);

/\*\*

\* @details 创建长度为length的双向链表,节点data不进行赋值.

\* @param length:双向链表的长度.

\* @return void.

\*/

void createLinkListWithoutForValue(int length);

/\*\*

\* @details 创建长度为length的双向链表,节点data进行赋值为0.

\* @param length:双向链表的长度.

\* @return void.

\*/

void createLinkListWithForValue(int length);

/\*\*

\* @details 在position位置后面插入节点.

\* @param item:插入的数据.

\* @param position:插入的位置.

\* @return void.

\*/

void insert(int item, int position);

/\*\*

\* @details 删除position位置的节点.

\* @param position:删除的位置.

\* @return void.

\*/

void erase(int position);

private:

//头指针.

node \*headPoint;

//链表长度.

int linkListLength;

//大整数正负标记.

int flag;

};

/\*\*

\* @details 输出运算符重载.

\* @param ostream &:输出对象, linkList &:待输出对象.

\* @return ostream &:待输出对象.

\*/

ostream & operator<<(ostream &, linkList &);

#endif //OPERATION\_OF\_LARGE\_INTEGERS\_LINKLIST\_H

===============================================================================

LinkList.cpp

#include "LinkList.h"

node \* linkList::returnHeadPoint() {

return headPoint;

}

int & linkList::returnLinkListLength() {

return linkListLength;

}

int & linkList::returnLinkListFlag() {

return flag;

}

int linkList::linkListSize() const {

return this->linkListLength;//直接返回linkListLength成员.

}

int linkList::linkListFlag() const {

return this->flag;//直接返回flag成员.

}

linkList::linkList() {

node \*headNode = new node();//创建头节点.

headNode->prev = headNode;//头节点前驱指向自己.

headNode->next = headNode;//头节点后继指向自己.

headPoint = headNode;//头指针指向头节点.

linkListLength = 0;//双向链表长度为0.

flag = 0;//正负标记设置为0.

}

linkList::linkList(const linkList &object) {

if (object.linkListSize() == 0) {//双向链表为空链表,链表中仅有头节点.

node \*headNode = new node();//创建头节点.

headNode->prev = headNode;//头节点前驱指向自己.

headNode->next = headNode;//头节点后继指向自己.

headPoint = headNode;//头指针指向头节点.

linkListLength = 0;//双向链表长度为0.

flag = 0;//正负标记设置为0.

} else {//双向链表非空.

//头节点的建立.

node \*headNode = new node();

headNode->prev = headNode;

headNode->next = headNode;

headPoint = headNode;

linkListLength = 0;

node \*tempPoint = object.headPoint->next;//第一个节点.

/\*\*

\* frontNode : 指向第一个节点的指针.

\* behindNode : 记录目前的链的最后一个节点.

\* newNode : 新创建的待加入的指针.

\*/

node \*frontNode = new node();//指向第一个节点的指针,第一个节点的深拷贝.

frontNode->data = tempPoint->data;

node \*behindNode = frontNode;//记录目前的链的最后一个节点.

tempPoint = tempPoint->next;

int size = object.linkListSize();//链表长度,即应当复制的节点数量.

int index = 1;//已复制的节点数量.

while (index < size) {

node \*newNode = new node();//复制新节点.

behindNode->next = newNode;

newNode->prev = behindNode;

behindNode = behindNode->next;//更新最后一个节点为新节点.

behindNode->data = tempPoint->data;

tempPoint = tempPoint->next;

++index;

}

headNode->next = frontNode;//将新链加到头节点后.

frontNode->prev = headNode;

behindNode->next = headNode;//将最后一个节点指向头节点,构成循环链.

headNode->prev = behindNode;

linkListLength = size;

flag = object.flag;//正负标记设置为object的flag.

}

}

linkList & linkList::operator=(const linkList &rightHandObject) {

if (this == &rightHandObject) {//判断进行自我赋值.

return \*this;

} else {

//双向链表为空链表,链表中仅有头节点.

if (rightHandObject.linkListSize() == 0) {

node \*headNode = new node();//创建头节点.

headNode->prev = headNode;//头节点前驱指向自己.

headNode->next = headNode;//头节点后继指向自己.

headPoint = headNode;//头指针指向头节点.

linkListLength = 0;//双向链表长度为0.

flag = 0;//正负标记设置为0.

} else {//双向链表非空.

//头节点的建立.

node \*headNode = new node();

headNode->prev = headNode;

headNode->next = headNode;

headPoint = headNode;

linkListLength = 0;

node \*tempPoint = rightHandObject.headPoint->next;//第一个节点.

/\*\*

\* frontNode : 指向第一个节点的指针.

\* behindNode : 记录目前的链的最后一个节点.

\* newNode : 新创建的待加入的指针.

\*/

//指向第一个节点的指针,第一个节点的深拷贝.

node \*frontNode = new node();

frontNode->data = tempPoint->data;

node \*behindNode = frontNode;//记录目前的链的最后一个节点.

tempPoint = tempPoint->next;

//链表长度,即应当复制的节点数量.

int size = rightHandObject.linkListSize();

int index = 1;//已复制的节点数量.

while (index < size) {

node \*newNode = new node();//复制新节点.

behindNode->next = newNode;

newNode->prev = behindNode;

behindNode = behindNode->next;//更新最后一个节点为新节点.

behindNode->data = tempPoint->data;

tempPoint = tempPoint->next;

++index;

}

headNode->next = frontNode;//将新链加到头节点后.

frontNode->prev = headNode;

behindNode->next = headNode;//将最后一个节点指向头节点,构成循环链.

headNode->prev = behindNode;

linkListLength = size;

flag = rightHandObject.flag;//正负标记设置为rightHandObject的flag.

}

return \*this;

}

}

linkList::~linkList() {

node \*tempPoint = headPoint;

for (int i = 0; i <= this->linkListLength; ++i) {

node \*deletePoint = tempPoint;

tempPoint = tempPoint->next;

delete deletePoint;

}

}

void linkList::display(ostream &out) {

if (flag == -1) {

out << '-';

}

node \*tempPoint = headPoint->next;

for (int i = 0; i < this->linkListLength; ++i) {//依个遍历.

out << tempPoint->data;

tempPoint = tempPoint->next;

}

cout << endl << "digit:" << linkListLength;

}

void linkList::createLinkListWithoutForValue(int length) {

//头节点的建立.

node \*headNode = new node();

headNode->prev = headNode;

headNode->next = headNode;

headPoint = headNode;

linkListLength = 0;

node \*frontNode = new node();//指向第一个节点的指针,第一个节点的创建.

node \*behindNode = frontNode;//记录目前的链的最后一个节点.

int number = 1;//已创建的新节点数量.

while (number < length) {

node \*newNode = new node();

behindNode->next = newNode;

newNode->prev = behindNode;

behindNode = behindNode->next;

++number;

}

headNode->next = frontNode;//头指针指向第一个节点.

frontNode->prev = headNode;

behindNode->next = headNode;//将最后一个节点指向头节点,构成循环链.

headNode->prev = behindNode;

linkListLength = length;

flag = 0;//正负标记设置为0.

}

void linkList::createLinkListWithForValue(int length) {

//头节点的建立.

node \*headNode = new node(0, nullptr, nullptr);

headNode->prev = headNode;

headNode->next = headNode;

headPoint = headNode;

linkListLength = 0;

//指向第一个节点的指针,第一个节点的创建.

node \*frontNode = new node(0, nullptr, nullptr);

node \*behindNode = frontNode;//记录目前的链的最后一个节点.

int number = 1;//已创建的新节点数量.

while (number < length) {

node \*newNode = new node(0, nullptr, nullptr);

behindNode->next = newNode;

newNode->prev = behindNode;

behindNode = behindNode->next;

++number;

}

headNode->next = frontNode;//头指针指向第一个节点.

frontNode->prev = headNode;

behindNode->next = headNode;//将最后一个节点指向头节点,构成循环链.

headNode->prev = behindNode;

linkListLength = length;

flag = 0;//正负标记设置为0.

}

void linkList::insert(int item, int position) {

node \*newPoint = new node(item, nullptr);//待插入的新节点.

node \*prePoint = headPoint;

node \*point = prePoint->next;

for (int i = 0; i < position; ++i) {

prePoint = prePoint->next;

point = point->next;

}

newPoint->next = prePoint->next;

prePoint->next = newPoint;

point->prev = newPoint;

newPoint->prev = prePoint;

linkListLength += 1;//长度自增1.

}

void linkList::erase(int position) {

node \*prePoint = headPoint;

node \*point = prePoint->next;

node \*behindPoint = point->next;

for (int i = 1; i < position; ++i) {

prePoint = prePoint->next;//位置前.

point = point->next;//删除节点的位置.

behindPoint = behindPoint->next;//位置后.

}

prePoint->next = behindPoint;

behindPoint->prev = prePoint;

delete point;

linkListLength -= 1;//长度自减1.

}

ostream & operator<<(ostream &out, linkList &printList) {

printList.display(out);//调用display函数.

return out;

}

===============================================================================

BigInteger.h

#ifndef OPERATION\_OF\_LARGE\_INTEGERS\_BIGINTEGER\_H

#define OPERATION\_OF\_LARGE\_INTEGERS\_BIGINTEGER\_H

#include <iostream>

#include <string>

#include <fstream>

#include "LinkList.h"

#include "Node.h"

using namespace std;

//大整数类.

class bigInteger {

public:

/\*\*

\* @details 将大整数（链表）转换为字符串.

\* @param list.

\* @return string:返回一个字符串.

\*/

static string translateToString(linkList &list);

/\*\*

\* @details 将数据写入文件.

\* @param data:待写入的数据.

\* @param url:文件路径.

\* @return void.

\*/

static void writeFile(const string& data, const string& url);

/\*\*

\* @details 大整数大小比较.

\* @param list1:链表1.

\* @param list2:链表2.

\* @return int:返回1代表list1>list2,-1代表list1<list2,0代表list1=list2.

\*/

static int comparisonOfSize(linkList &list1, linkList &list2);

/\*\*

\* @details 链表长度比较.

\* @param list1:链表1.

\* @param list2:链表2.

\* @return bool:若链表1大于链表2,返回true,反之返回false.

\*/

static bool compareListLength(linkList &list1, linkList &list2);

/\*\*

\* @details 不涉及文件的输入函数,将操作数存入链表.

\* @param object1:用于存储大整数1.

\* @param object2:用于存储大整数2.

\* @return void.

\*/

static void inputListWithoutFile(linkList &object1, linkList &object2);

/\*\*

\* @details 涉及文件的操作函数,十进制.

\* @param object1:用于存储大整数1.

\* @param object2:用于存储大整数2.

\* @param choice:运算种类.

\* @return void.

\*/

static void listOperateWithFile\_decimalSystem(linkList &object1,

linkList &object2, char choice);

/\*\*

\* @details 涉及文件的操作函数,二进制.

\* @param object1:用于存储大整数1.

\* @param object2:用于存储大整数2.

\* @param choice:运算种类.

\*/

static void listOperateWithFile\_binarySystem(linkList &object1,

linkList &object2, char choice);

/\*\*

\* @details 加法辅助函数.

\* @param object1:操作数1，即大整数1.

\* @param object2:操作数2，即大整数2.

\* @param base:进制.

\* @return linkList.

\*/

static linkList additionAuxiliaryFunction(linkList &object1,

linkList &object2, int base);

/\*\*

\* @details 减法辅助函数.

\* @param object1:操作数1，即大整数1.

\* @param object2:操作数2，即大整数2.

\* @param base:进制.

\* @return linkList.

\*/

static linkList subtractionAuxiliaryFunction(linkList &object1,

linkList &object2, int base);

/\*\*

\* @details 乘法辅助函数.

\* @param object1:操作数1，即大整数1.

\* @param object2:操作数2，即大整数2.

\* @param base:进制.

\* @return linkList.

\*/

static linkList multiplicationAuxiliaryFunction(linkList &object1,

linkList &object2, int base);

/\*\*

\* @details 乘法子函数.

\* @param operator1:操作数1.

\* @param operator2:操作数2.

\* @param base:进制.

\* @return linkList.

\*/

static linkList divisionFunction(linkList &operator1,

linkList &operator2, int base);

/\*\*

\* @details 除法辅助函数.

\* @param object1:操作数1，即大整数1.

\* @param object2:操作数2，即大整数2.

\* @param base:进制.

\* @return linkList.

\*/

static linkList divisionAuxiliaryFunction(linkList &object1,

linkList &object2, int base);

/\*\*

\* @details 十进制加法函数,可正可负.

\* @param operation1:操作数1，即大整数1.

\* @param operation2:操作数2，即大整数2.

\* @param base:进制.

\* @return void.

\*/

static void addition\_bigInteger(linkList &operation1,

linkList &operation2, int base);

/\*\*

\* @details 二进制加法函数.

\* @param object1:操作数1，即大整数1.

\* @param object2:操作数2，即大整数2.

\* @return void.

\*/

static void addition\_bigInteger\_binarySystem(linkList &object1,

linkList &object2);

/\*\*

\* @details 十进制减法函数,可正可负.

\* @param operation1:操作数1，即大整数1.

\* @param operation2:操作数2，即大整数2.

\* @param base:进制.

\* @return void.

\*/

static void subtraction\_bigInteger(linkList &operation1,

linkList &operation2, int base);

/\*\*

\* @details 二进制减法函数.

\* @param object1:操作数1，即大整数1.

\* @param object2:操作数2，即大整数2.

\* @return void.

\*/

static void subtraction\_bigInteger\_binarySystem(linkList &object1,

linkList &object2);

/\*\*

\* @details 十进制乘法函数,可正可负.

\* @param operation1:操作数1，即大整数1.

\* @param operation2:操作数2，即大整数2.

\* @param base:进制.

\* @return void.

\*/

static void multiplication\_bigInteger(linkList &operation1,

linkList &operation2, int base);

/\*\*

\* @details 二进制乘法函数.

\* @param object1:操作数1，即大整数1.

\* @param object2:操作数2，即大整数2.

\* @return void.

\*/

static void multiplication\_bigInteger\_binarySystem(

linkList &object1, linkList &object2);

/\*\*

\* @details 十进制除法函数,可正可负.

\* @param operation1:操作数1，即大整数1.

\* @param operation2:操作数2，即大整数2.

\* @param base:进制.

\* @return void.

\*/

static void division\_bigInteger(linkList &operation1,

linkList &operation2, int base);

/\*\*

\* @details 二进制除法函数.

\* @param object1:操作数1，即大整数1.

\* @param object2:操作数2，即大整数2.

\* @return void.

\*/

static void division\_bigInteger\_binarySystem(linkList &object1,

linkList &object2);

/\*\*

\* @details 十、二进制指数运算函数.

\* @param operation1:操作数1，即大整数1.

\* @param operation2:操作数2，即大整数2.

\* @param base:进制.

\* @return void.

\*/

static void exponentialCalculation\_bigInteger(linkList &operation1,

linkList &operation2, int base);

};

#endif //Operation\_OF\_LARGE\_INTEGERS\_BIGINTEGER\_H

===============================================================================

BigInteger.cpp

#include "BigInteger.h"

string bigInteger::translateToString(linkList &list) {

node \*back;

int length = list.linkListSize();

string result(length, 0);//创建长度为l,初始化为0的字符串.

back = list.returnHeadPoint()->prev;

for (int i = length - 1; i >= 0; i--) {//循环将链表数据输入到字符串中.

result[i] = back->data + '0';

back = back->prev;

}

if (list.linkListFlag() == -1) {

result = "-" + result;

}

return result;

}

void bigInteger::writeFile(const string& data, const string& url) {

ofstream fileStream;//文件流对象.

fileStream.open(url, ios::out | ios::app);

//文件不存在.

if (!fileStream.is\_open()) {

fileStream.close();

return;

}

fileStream << data << endl;

fileStream.close();

}

int bigInteger::comparisonOfSize(linkList &list1, linkList &list2) {

int length1 = list1.returnLinkListLength();

int length2 = list2.returnLinkListLength();

int flag = 0;

if (length1 > length2) {//list1长度大于list2长度.

flag = 1;

} else if (length1 == length2) {//长度相等.

node \*point1 = list1.returnHeadPoint()->next;

node \*point2 = list2.returnHeadPoint()->next;

for (int i = 0; i < length1; i++) {//从高位开始比较大小.

if (point1->data > point2->data) {

flag = 1;

break;

} else if (point1->data < point2->data) {

flag = -1;

break;

}

point1 = point1->next;

point2 = point2->next;

}

} else {//list1长度小于list2长度.

flag = -1;

}

return flag;

}

bool bigInteger::compareListLength(linkList &list1, linkList &list2) {

int listLength1 = list1.linkListSize();

int listLength2 = list2.linkListSize();

return listLength1 > listLength2;

}

void bigInteger::inputListWithoutFile(linkList &object1, linkList &object2) {

int length;

node \*back;

int flag;//符号标记.

//第一个长整数的输入及存储.

cout << "请输入第一个操作数：";

string string1;

cin >> string1;

flag = 1;

if (string1[0] == '-') {

string1 = string1.substr(1);

flag = -1;

}

length = string1.length();

object1.createLinkListWithoutForValue(length);//创建长度为length1的双向循环链表,赋值给object1.

back = object1.returnHeadPoint()->prev;

object1.returnLinkListFlag() = flag;

//将大整数存储到链表内.

for (int i = length - 1; i >= 0; --i) {

back->data = string1[i] - '0';

back = back->prev;

}

//第二个长整数的输入及存储.

cout << "请输入第二个操作数：";

string string2;

cin >> string2;

flag = 1;

if (string2[0] == '-') {

string2 = string2.substr(1);

flag = -1;

}

length = string2.length();

object2.createLinkListWithoutForValue(length);//创建长度为length1的双向循环链表,赋值给object2.

back = object2.returnHeadPoint()->prev;

object2.returnLinkListFlag() = flag;

//将大整数存储到链表内.

for (int i = length - 1; i >= 0; --i) {

back->data = string2[i] - '0';

back = back->prev;

}

}

void bigInteger::listOperateWithFile\_decimalSystem(linkList &object1, linkList &object2, char choice) {

int flag;//符号标记.

ifstream fileStream;//文件流对象.

fileStream.open(R"(C:\Users\15590\Desktop\Operation of large integers\file\file\_decimalSystem.txt)", ios::in);//打开文件.

//文件不存在.

if (!fileStream.is\_open()) {

fileStream.close();

return;

}

//文件存在,但为空文件.

char testData;

fileStream >> testData;

if (fileStream.eof()) {

cout << "文件为空！\n";

fileStream.close();

return;

}

//若文件存在，并且数据存在.

string string1, string2;

int length;

node \*back;

while (fileStream >> string1 >> string2) {

flag = 1;

if (string1[0] == '-') {

string1 = string1.substr(1);

flag = -1;

}

length = string1.length();

object1.createLinkListWithoutForValue(length);//创建长度为length1的双向循环链表,赋值给object1.

back = object1.returnHeadPoint()->prev;

object1.returnLinkListFlag() = flag;

//将大整数存储到链表内.

for (int i = length - 1; i >= 0; --i) {

back->data = string1[i] - '0';

back = back->prev;

}

flag = 1;

if (string2[0] == '-') {

string2 = string1.substr(1);

flag = -1;

}

length = string2.length();

object2.createLinkListWithoutForValue(length);//创建长度为length1的双向循环链表,赋值给object2.

back = object2.returnHeadPoint()->prev;

object2.returnLinkListFlag() = flag;

//将大整数存储到链表内.

for (int i = length - 1; i >= 0; --i) {

back->data = string2[i] - '0';

back = back->prev;

}

cout << "list1:" << object1 << endl;

cout << "list2:" << object2 << endl;

switch (choice) {

case '+': {

cout << "加法 —>>> (list1 + list2) = ";

addition\_bigInteger(object1, object2, 10);

break;

}

case '-': {

cout << "减法 —>>> (list1 - list2) = ";

subtraction\_bigInteger(object1, object2, 10);

break;

}

case '\*': {

cout << "乘法 —>>> (list1 \* list2) = ";

multiplication\_bigInteger(object1, object2, 10);

break;

}

case '/': {

cout << "除法 —>>> (list1 / list2) = ";

division\_bigInteger(object1, object2, 10);

break;

}

default: {

cout << "指数运算 —>>> (list1 ^ list2) = ";

exponentialCalculation\_bigInteger(object1, object2, 10);

break;

}

}

}

fileStream.close();

}

void bigInteger::listOperateWithFile\_binarySystem(linkList &object1, linkList &object2, char choice) {

int flag;//符号标记.

ifstream fileStream;//文件流对象.

fileStream.open(R"(C:\Users\15590\Desktop\Operation of large integers\file\file\_binarySystem.txt)", ios::in);//打开文件.

//文件不存在.

if (!fileStream.is\_open()) {

fileStream.close();

return;

}

//文件存在,但为空文件.

char testData;

fileStream >> testData;

if (fileStream.eof()) {

cout << "文件为空！\n";

fileStream.close();

return;

}

//若文件存在，并且数据存在.

string string1, string2;

int length;

node \*back;

while (fileStream >> string1 >> string2) {

flag = 1;

if (string1[0] == '-') {

string1 = string1.substr(1);

flag = -1;

}

length = string1.length();

object1.createLinkListWithoutForValue(length);//创建长度为length的双向循环链表,赋值给object1.

back = object1.returnHeadPoint()->prev;

object1.returnLinkListFlag() = flag;

//将大整数存储到链表内.

for (int i = length - 1; i >= 0; --i) {

back->data = string1[i] - '0';

back = back->prev;

}

flag = 1;

if (string2[0] == '-') {

string2 = string1.substr(1);

flag = -1;

}

length = string2.length();

object2.createLinkListWithoutForValue(length);//创建长度为length的双向循环链表,赋值给object2.

back = object2.returnHeadPoint()->prev;

object2.returnLinkListFlag() = flag;

//将大整数存储到链表内.

for (int i = length - 1; i >= 0; --i) {

back->data = string2[i] - '0';

back = back->prev;

}

cout << "list1:" << object1 << endl;

cout << "list2:" << object2 << endl;

switch (choice) {

case '+': {

cout << "加法 —>>> (list1 + list2) = ";

addition\_bigInteger\_binarySystem(object1, object2);

break;

}

case '-': {

cout << "减法 —>>> (list1 - list2) = ";

subtraction\_bigInteger\_binarySystem(object1, object2);

break;

}

case '\*': {

cout << "乘法 —>>> (list1 \* list2) = ";

multiplication\_bigInteger\_binarySystem(object1, object2);

break;

}

case '/': {

cout << "除法 —>>> (list1 / list2) = ";

division\_bigInteger\_binarySystem(object1, object2);

break;

}

default: {

cout << "指数运算 —>>> (list1 ^ list2) = ";

exponentialCalculation\_bigInteger(object1, object2, 2);

break;

}

}

}

fileStream.close();

}

linkList bigInteger::additionAuxiliaryFunction(linkList &object1, linkList &object2, int base) {

linkList result;

bool lengthJudge = compareListLength(object1, object2);//判断传入的两个操作数的长度大小.

if (lengthJudge) {//object1的长度 > object2的长度.

int objectLength1 = object1.linkListSize();

int objectLength2 = object2.linkListSize();

int quotientRecord = 0;//quotient记录进位.

node \*tempPoint1 = object1.returnHeadPoint()->prev;//tempPoint1指向object1表尾节点.

node \*tempPoint2 = object2.returnHeadPoint()->prev;//tempPoint2指向object2表尾节点.

result.createLinkListWithoutForValue(objectLength1);//创建结果链表长度为object1和object2中长度较大者.

node \*resultTempPoint = result.returnHeadPoint()->prev;//resultTempPoint指向result表尾节点.

for (int i = 0; i < objectLength2; ++i) {//做object1和object2中长度较小者的长度次循环.

int sum = tempPoint1->data + tempPoint2->data + quotientRecord;//sum记录每位的和.

int valueForBit = sum % base;//valueForBit为每位应当记录的的结果.

quotientRecord = sum / base;//quotient记录进位.

tempPoint1 = tempPoint1->prev;

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = valueForBit;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

}

for (int i = objectLength2; i < objectLength1; ++i) {//做相差长度次循环.

int sum = tempPoint1->data + quotientRecord;

int valueForBit = sum % base;//valueForBit为每位应当记录的的结果.

quotientRecord = sum / base;//quotient记录进位.

tempPoint1 = tempPoint1->prev;

resultTempPoint->data = valueForBit;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

}

if (quotientRecord == 1) {

result.insert(1, 0);

}

} else {//object1的长度 <= object2的长度.

int objectLength1 = object1.linkListSize();

int objectLength2 = object2.linkListSize();

int quotientRecord = 0;//quotient记录进位.

node \*tempPoint1 = object1.returnHeadPoint()->prev;//tempPoint1指向object1表尾节点.

node \*tempPoint2 = object2.returnHeadPoint()->prev;//tempPoint2指向object2表尾节点.

result.createLinkListWithoutForValue(objectLength2);//创建结果链表长度为object1和object2中长度较大者.

node \*resultTempPoint = result.returnHeadPoint()->prev;//resultTempPoint指向result表尾节点.

for (int i = 0; i < objectLength1; ++i) {//做object1和object2中长度较小者的长度次循环.

int sum = tempPoint1->data + tempPoint2->data + quotientRecord;//sum记录每位的和.

int valueForBit = sum % base;//valueForBit为每位应当记录的的结果.

quotientRecord = sum / base;//quotient记录进位.

tempPoint1 = tempPoint1->prev;

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = valueForBit;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

}

for (int i = objectLength1; i < objectLength2; ++i) {//做相差长度次循环.

int sum = tempPoint2->data + quotientRecord;

int valueForBit = sum % base;//valueForBit为每位应当记录的的结果.

quotientRecord = sum / base;//quotient记录进位.

tempPoint1 = tempPoint1->prev;

resultTempPoint->data = valueForBit;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

}

if (quotientRecord == 1) {

result.insert(1, 0);

}

}

return result;

}

linkList bigInteger::subtractionAuxiliaryFunction(linkList &object1, linkList &object2, int base) {

linkList result;

int objectLength1 = object1.linkListSize();

int objectLength2 = object2.linkListSize();

if (objectLength1 > objectLength2) {//object1的长度 > object2的长度.

int subRecord = 0;//subRecord记录借位.

int sum = 0;

node \*tempPoint1 = object1.returnHeadPoint()->prev;//tempPoint1指向object1表尾节点.

node \*tempPoint2 = object2.returnHeadPoint()->prev;//tempPoint2指向object2表尾节点.

result.createLinkListWithoutForValue(objectLength1);//创建结果链表长度为object1和object2中长度较大者.

node \*resultTempPoint = result.returnHeadPoint()->prev;//resultTempPoint指向result表尾节点.

for (int i = 0; i < objectLength2; ++i) {//做object1和object2中长度较小者的长度次循环.

if ((tempPoint1->data - subRecord) < tempPoint2->data) {//需要借位.

sum = tempPoint1->data + base - tempPoint2->data - subRecord;//sum记录每位的差.

subRecord = 1;//subRecord记录借位.

} else {//不需要借位.

sum = tempPoint1->data - tempPoint2->data - subRecord;//sum记录每位的差.

subRecord = 0;//subRecord记录借位.

}

tempPoint1 = tempPoint1->prev;

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = sum;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

}

for (int i = objectLength2; i < objectLength1; ++i) {//做相差长度次循环.

if (tempPoint1->data < subRecord) {//若此位为0.

sum = tempPoint1->data - subRecord + base;//从前一位借1,sum记录每位相减的差.

subRecord = 1;//sub记录借位.

} else {//若此位不为为0.

sum = tempPoint1->data - subRecord;

subRecord = 0;//sub记录借位.

}

tempPoint1 = tempPoint1->prev;

resultTempPoint->data = sum;

resultTempPoint=resultTempPoint->prev;

}

if (sum == 0) {

result.erase(1);

}

} else if (objectLength1 < objectLength2) {//object1的长度 < object2的长度.

int subRecord = 0;//subRecord记录借位.

int sum = 0;

node \*tempPoint1 = object1.returnHeadPoint()->prev;//tempPoint1指向object1表尾节点.

node \*tempPoint2 = object2.returnHeadPoint()->prev;//tempPoint2指向object2表尾节点.

result.createLinkListWithoutForValue(objectLength2);//创建结果链表长度为object1和object2中长度较大者.

node \*resultTempPoint = result.returnHeadPoint()->prev;//resultTempPoint指向result表尾节点.

for (int i = 0; i < objectLength1; ++i) {//做object1和object2中长度较小者的长度次循环.

if ((tempPoint2->data - subRecord) < tempPoint1->data) {//需要借位.

sum = tempPoint2->data + base - tempPoint1->data - subRecord;//sum记录每位的差.

subRecord = 1;//subRecord记录借位.

} else {//不需要借位.

sum = tempPoint2->data - tempPoint1->data - subRecord;//sum记录每位的差.

subRecord = 0;//subRecord记录借位.

}

tempPoint1 = tempPoint1->prev;

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = sum;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

}

for (int i = objectLength1; i < objectLength2; ++i) {//做相差长度次循环.

if (tempPoint2->data < subRecord) {//若此位为0.

sum = tempPoint2->data - subRecord + base;//从前一位借1,sum记录每位相减的差.

subRecord = 1;//sub记录借位.

} else {//若此位不为为0.

sum = tempPoint2->data - subRecord;

subRecord = 0;//sub记录借位.

}

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = sum;

resultTempPoint=resultTempPoint->prev;

}

if (sum == 0) {

result.erase(1);

}

} else {//object1的长度 = object2的长度.

int sizeMark = 0;

node \*tempPoint1 = object1.returnHeadPoint()->prev;//tempPoint1指向object1表尾节点.

node \*tempPoint2 = object2.returnHeadPoint()->prev;//tempPoint2指向object2表尾节点.

node \*markPoint1 = object1.returnHeadPoint()->next;//tempPoint1指向object1第一个节点.

node \*markPoint2 = object2.returnHeadPoint()->next;//tempPoint2指向object2第一个节点.

//循环用于比较两个长整数大小.

for (int i = 0; i < objectLength1; ++i) {

if (markPoint1->data > markPoint2->data) {

sizeMark = 1;//第一个长整数大,m赋值为1.

break;

} else if (markPoint1->data < markPoint2->data) {

sizeMark = 2;//第二个长整数大,m赋值为2.

break;

}

markPoint1 = markPoint1->next;

markPoint2 = markPoint2->next;

}

if (sizeMark == 0) {//两个长整数相等.

result.createLinkListWithoutForValue(1);

result.returnHeadPoint()->next->data = 0;//最后计算结果为0.

}

if (sizeMark == 1) {//第一个长整数大.

result.createLinkListWithoutForValue(objectLength1);//链表长度为二者之中任意长度.

node \*resultTempPoint = result.returnHeadPoint()->prev;//resultTempPoint指向result表尾节点.

int subRecord = 0;//subRecord记录借位.

int sum = 0;

for (int i = 0; i < objectLength1; i++) {//循环位数次.

if ((tempPoint1->data - subRecord) < tempPoint2->data) {

sum = tempPoint1->data + base - tempPoint2->data - subRecord;//从前一位借1,sum记录每位相减的差.

subRecord = 1;//subRecord记录借位.

} else {

sum = tempPoint1->data - tempPoint2->data - subRecord;//直接相减.

subRecord = 0;//sub记录借位.

}

tempPoint1 = tempPoint1->prev;

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = sum;//将结果记入链表.

resultTempPoint = resultTempPoint->prev;

}

if (sum == 0) {//若最大数位为0.

result.erase(1);//将第一个节点,即数据0删除.

}

node \*temp1 = result.returnHeadPoint()->next;

node \*temp2 = temp1->next;

node \*temp3 = result.returnHeadPoint();

while (temp1->data == 0) {//存在0001的情况,循环用于删除多余0.

delete temp1;//删除该节点.

temp1 = temp2;

temp2 = temp2->next;

result.returnLinkListLength() = result.returnLinkListLength() - 1;//链表长度减1.

}

temp3->next = temp1;//删除节点之后的连接操作.

temp1->prev = temp3;

} else if (sizeMark == 2) {//第二个长整数大

result.createLinkListWithoutForValue(objectLength1);//链表长度为二者之中任意长度.

node \*resultTempPoint = result.returnHeadPoint()->prev;//resultTempPoint指向result表尾节点.

int subRecord = 0;//subRecord记录借位.

int sum = 0;

for (int i = 0; i < objectLength1; i++) {//循环位数次.

if ((tempPoint2->data - subRecord) < tempPoint1->data) {

sum = tempPoint2->data + base - tempPoint1->data - subRecord;//从前一位借1,sum记录每位相减的差.

subRecord = 1;//sub记录借位.

} else {

sum = tempPoint2->data - tempPoint1->data - subRecord;//直接相减.

subRecord = 0;//sub记录借位.

}

tempPoint1 = tempPoint1->prev;

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = sum;//将结果记入链表.

resultTempPoint = resultTempPoint->prev;

}

if (sum == 0) {//若最大数位为0.

result.erase(1);//将第一个节点,即数据0删除.

}

node \*temp1 = result.returnHeadPoint()->next;

node \*temp2 = temp1->next;

node \*temp3 = result.returnHeadPoint();

while (temp1->data == 0) {//存在0001的情况,循环用于删除多余0.

delete temp1;//删除该节点.

temp1 = temp2;

temp2 = temp2->next;

result.returnLinkListLength() = result.returnLinkListLength() - 1;//链表长度减1.

}

temp3->next = temp1;//删除节点之后的连接操作.

temp1->prev = temp3;

}

}

return result;

}

linkList bigInteger::multiplicationAuxiliaryFunction(linkList &object1, linkList &object2, int base) {

linkList result;

int objectLength1 = object1.linkListSize();

int objectLength2 = object2.linkListSize();

//judgePoint1和judgePoint2用来判断两个链表是否为0链表.

node \*judgePoint1 = object1.returnHeadPoint()->next;

node \*judgePoint2 = object2.returnHeadPoint()->next;

node \*tempPoint2 = object2.returnHeadPoint()->prev;//tempPoint2指向object2表尾节点.

//两个大整数其中有一个为0.

if (judgePoint1->data == 0 || judgePoint2->data == 0) {

result.insert(0,0);//相乘结果为0.

} else {

result.createLinkListWithForValue(objectLength1 + objectLength2);//积链表长度为两个链表长度之和,并全部初始化.

//markPoint负责把每次相乘的结果记入链表,tempPoint往前对位.

node \*tempPoint = result.returnHeadPoint()->prev;

node \*markPoint = tempPoint;

for (int i = 0; i < objectLength2; ++i) {

if (tempPoint2->data == 0) {

tempPoint2 = tempPoint2->prev;

tempPoint = tempPoint->prev;

markPoint = tempPoint;

continue;

}

int temp = 0;

node \*tempPoint1 = object1.returnHeadPoint()->prev;//tempPoint1指向object1表尾节点.

for (int j = 0; j < objectLength1; ++j) {

//每次需要相加三部分数字.

temp = temp + (tempPoint2->data) \* (tempPoint1->data) + markPoint->data;

markPoint->data = temp % base;//结果.

temp = temp / base;//进位.

tempPoint1 = tempPoint1->prev;

markPoint = markPoint->prev;

}

markPoint->data = markPoint->data + temp;//进位后的结果记入链表.

tempPoint2 = tempPoint2->prev;

tempPoint = tempPoint->prev;

markPoint = tempPoint;

}

if (result.returnHeadPoint()->next->data == 0)//最大数位为0.

result.erase(1);//删除0.

}

return result;

}

linkList bigInteger::divisionFunction(linkList &operator1, linkList &operator2, int base) {

int l1=operator1.linkListSize();

int l2=operator2.linkListSize();

node\*ptr1=operator1.returnHeadPoint()->prev;//指向链表尾

node\*ptr2=operator2.returnHeadPoint()->prev;//指向链表尾

linkList d;

d.createLinkListWithoutForValue(l1);//较长者的长度

node\*ptr3=d.returnHeadPoint()->prev;

int sum,sub=0;

for(int i=0;i<l2;i++)

{

if((ptr1->data-sub)<ptr2->data)

{

sum=ptr1->data+10-ptr2->data-sub;//借位后相减

sub=1;//记录借位

}

else

{

sum=ptr1->data-ptr2->data-sub;//直接相减

sub=0;//记录借位

}

ptr1=ptr1->prev;

ptr2=ptr2->prev;

ptr3->data=sum;//将结果记入链表中

ptr3=ptr3->prev;

}

for(int k=l2;k<l1;k++)//差位次循环

{

if(ptr1->data<sub)//小于0

{

sum=ptr1->data-sub+10;//借位相减

sub=1;//记录借位

}

else

{

sum=ptr1->data-sub;//直接相减

sub=0;//借位

}

ptr1=ptr1->prev;

ptr3->data=sum;//将结果记入链表中

ptr3=ptr3->prev;

}

node\*ptr4=d.returnHeadPoint()->next;

node\*ptr5=ptr4->next;

node\*ptr6=d.returnHeadPoint();

while(ptr4->data==0&&d.linkListSize()>1)//存在0001情况,循环用于删除多余0

{

delete ptr4;//删除多余0

ptr4=ptr5;

ptr5=ptr5->next;

d.returnLinkListLength()=d.returnLinkListLength()-1;//链表长度减1

}

ptr6->next=ptr4;//删除节点之后的连接操作

ptr4->prev=ptr6;

return d;

}

linkList bigInteger::divisionAuxiliaryFunction(linkList &object1, linkList &object2, int base) {

linkList result;

if (comparisonOfSize(object1, object2) == -1) {//除数比被除数大的情况.

result.insert(0, 0);

} else {//除数和被除数相等或被除数比除数大的情况.

if (object2.returnHeadPoint()->next->data == 0) {//除数为0的情况.

cout << "Divide by 0, no result!" << endl;

} else if (object1.returnHeadPoint()->next->data == 0) {//被除数为0的情况.

result.insert(0, 0);

} else {//其余情况.

linkList tempList;

int objectLength1 = object1.linkListSize();

int objectLength2 = object2.linkListSize();

if ((objectLength2 - 1) != 0) {

tempList.createLinkListWithoutForValue(objectLength2 - 1);

node \*tempListPtr = tempList.returnHeadPoint()->next;

node \*ptr = object1.returnHeadPoint()->next;

for (int i = 0; i < objectLength2 - 1; ++i) {//循环赋值.

tempListPtr->data = ptr->data;

tempListPtr = tempListPtr->next;

ptr = ptr->next;

}

}

node \*markPtr = object1.returnHeadPoint()->next;

for (int i = 0; i < objectLength2 - 1; ++i) {//寻找object2最高位位于object1的位置.

markPtr = markPtr->next;

}

node \*erasePoint;

for (int i = objectLength2 - 1; i < objectLength1; ++i) {

int count = 0;//count记录相减次数.

erasePoint = tempList.returnHeadPoint()->next;

if (erasePoint->data == 0) {//相减之后差为0,将0后的数代替0.

node \*change = tempList.returnHeadPoint()->next;

change->data = markPtr->data;

} else {//继续插入下一位数.

tempList.insert(markPtr->data, tempList.linkListSize());

}

while ((comparisonOfSize(tempList, object2)) >= 0) {//比较两个数的大小.

tempList = subtractionAuxiliaryFunction(tempList, object2, base);//做连减操作.

count++;//记录连减次数.

}

markPtr = markPtr->next;

result.insert(count, result.linkListSize());//将每次连减的次数插入到result中.

}

if (result.returnHeadPoint()->next->data == 0) {//若最大数为0.

result.erase(1);

}

}

}

return result;

}

void bigInteger::addition\_bigInteger(linkList &operation1, linkList &operation2, int base) {

linkList result;

time\_t timeStart, timeEnd;//时间变量.

time(&timeStart);//设置起始时间.

if (operation1.linkListFlag() == -1) {//第一个加数为负数.

if (operation2.linkListFlag() == 1) {//第二个加数为正数.

if (comparisonOfSize(operation1, operation2) == 1) {//第一个数的绝对值大于第二个数.

result = subtractionAuxiliaryFunction(operation1, operation2, base);

result.returnLinkListFlag() = -1;

cout << result;

cout << endl;

} else {//第一个数的绝对值小于等于第二个数.

result = subtractionAuxiliaryFunction(operation2, operation1, base);

cout << result;

cout << endl;

}

} else {//第二个加数为负数或0.

result = additionAuxiliaryFunction(operation1, operation2, base);

result.returnLinkListFlag() = -1;

cout << result;

cout << endl;

}

} else {//第一个加数为正数或0.

if (operation2.linkListFlag() == -1) {//第二个加数为负数.

if (comparisonOfSize(operation1, operation2) == -1) {//第一个数的绝对值小于第二个数.

result = subtractionAuxiliaryFunction(operation2, operation1,base);

result.returnLinkListFlag() = -1;

cout << result;

cout << endl;

} else {//第一个数的绝对值大于等于第二个数.

result = subtractionAuxiliaryFunction(operation1, operation2, base);

cout << result;

cout << endl;

}

} else {//第二个加数为正数或0.

result = additionAuxiliaryFunction(operation1, operation2, base);

cout << result;

cout << endl;

}

}

time(&timeEnd);

cout << "time: " << timeEnd - timeStart << endl;

cout << "Do you want to save the results in a file:(1.Yes/2.No)";

int choice;

cin >> choice;

if (choice == 1) {

string str = translateToString(result);

writeFile(str, R"(C:\Users\15590\Desktop\Operation of large integers\file\result\_decimalSystem.txt)");

}

}

void bigInteger::addition\_bigInteger\_binarySystem(linkList &object1, linkList &object2) {

time\_t timeStart, timeEnd;//时间变量.

linkList result;

time(&timeStart);//设置起始时间.

bool lengthJudge = compareListLength(object1, object2);//判断传入的两个操作数的长度大小.

if (lengthJudge) {//object1的长度 > object2的长度.

int objectLength1 = object1.linkListSize();

int objectLength2 = object2.linkListSize();

int quotientRecord = 0;//quotient记录进位.

node \*tempPoint1 = object1.returnHeadPoint()->prev;//tempPoint1指向object1表尾节点.

node \*tempPoint2 = object2.returnHeadPoint()->prev;//tempPoint2指向object2表尾节点.

result.createLinkListWithoutForValue(objectLength1);//创建结果链表长度为object1和object2中长度较大者.

node \*resultTempPoint = result.returnHeadPoint()->prev;//resultTempPoint指向result表尾节点.

for (int i = 0; i < objectLength2; ++i) {//做object1和object2中长度较小者的长度次循环.

int sum = tempPoint1->data + tempPoint2->data + quotientRecord;//sum记录每位的和.

int valueForBit = sum % 2;//valueForBit为每位应当记录的的结果.

quotientRecord = sum / 2;//quotient记录进位.

tempPoint1 = tempPoint1->prev;

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = valueForBit;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

}

for (int i = objectLength2; i < objectLength1; ++i) {//做相差长度次循环.

int sum = tempPoint1->data + quotientRecord;

int valueForBit = sum % 2;//valueForBit为每位应当记录的的结果.

quotientRecord = sum / 2;//quotient记录进位.

tempPoint1 = tempPoint1->prev;

resultTempPoint->data = valueForBit;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

}

if (quotientRecord == 1) {

result.insert(1, 0);

}

} else {//object1的长度 <= object2的长度.

int objectLength1 = object1.linkListSize();

int objectLength2 = object2.linkListSize();

int quotientRecord = 0;//quotient记录进位.

node \*tempPoint1 = object1.returnHeadPoint()->prev;//tempPoint1指向object1表尾节点.

node \*tempPoint2 = object2.returnHeadPoint()->prev;//tempPoint2指向object2表尾节点.

result.createLinkListWithoutForValue(objectLength2);//创建结果链表长度为object1和object2中长度较大者.

node \*resultTempPoint = result.returnHeadPoint()->prev;//resultTempPoint指向result表尾节点.

for (int i = 0; i < objectLength1; ++i) {//做object1和object2中长度较小者的长度次循环.

int sum = tempPoint1->data + tempPoint2->data + quotientRecord;//sum记录每位的和.

int valueForBit = sum % 2;//valueForBit为每位应当记录的的结果.

quotientRecord = sum / 2;//quotient记录进位.

tempPoint1 = tempPoint1->prev;

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = valueForBit;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

}

for (int i = objectLength1; i < objectLength2; ++i) {//做相差长度次循环.

int sum = tempPoint2->data + quotientRecord;

int valueForBit = sum % 2;//valueForBit为每位应当记录的的结果.

quotientRecord = sum / 2;//quotient记录进位.

tempPoint1 = tempPoint1->prev;

resultTempPoint->data = valueForBit;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

}

if (quotientRecord == 1) {

result.insert(1, 0);

}

}

cout << result << endl;

time(&timeEnd);

cout << "time: " << timeEnd - timeStart << endl;

cout << "Do you want to save the results in a file:(1.Yes/2.No)";

int choice;

cin >> choice;

if (choice == 1) {

string str = translateToString(result);

writeFile(str, R"(C:\Users\15590\Desktop\Operation of large integers\file\result\_binarySystem.txt)");

}

}

void bigInteger::subtraction\_bigInteger(linkList &operation1, linkList &operation2, int base) {

linkList result;

time\_t timeStart, timeEnd;//时间变量.

time(&timeStart);//设置起始时间.

if (operation2.linkListFlag() == 0) {//第二个大整数为0.

result = operation1;

} else if (operation2.linkListFlag() == 1) {//第二个大整数为正数.

if (operation1.linkListFlag() == 1) {//第一个大整数为正数.

if (comparisonOfSize(operation1, operation2) == -1) {//第一个整数的绝对值小于第二个整数.

result = subtractionAuxiliaryFunction(operation2, operation1, base);

result.returnLinkListFlag() = -1;

cout << result;

cout << endl;

} else {//第一个整数的绝对值大于等于第二个整数.

result = subtractionAuxiliaryFunction(operation1, operation2, base);

cout << result;

cout << endl;

}

} else {//第一个大整数为负数或0.

result = additionAuxiliaryFunction(operation1, operation2, base);

result.returnLinkListFlag() = -1;

cout << result;

cout << endl;

}

} else {//第二个大整数为负数.

if (operation1.linkListFlag() == -1) {//第一个大整数为负数.

if (comparisonOfSize(operation1, operation2) == 1) {

result = subtractionAuxiliaryFunction(operation1, operation2, base);

result.returnLinkListFlag() = -1;

cout << result;

cout << endl;

} else {//第一个整数的绝对值小于等于第二个整数.

result = subtractionAuxiliaryFunction(operation2, operation1, base);

cout << result;

cout << endl;

}

} else {//第一个大整数为正数或0.

result = additionAuxiliaryFunction(operation1, operation2, base);

cout << result;

cout << endl;

}

}

time(&timeEnd);

cout << "time: " << timeEnd - timeStart << endl;

cout << "Do you want to save the results in a file:(1.Yes/2.No)";

int choice;

cin >> choice;

if (choice == 1) {

string str = translateToString(result);

writeFile(str, R"(C:\Users\15590\Desktop\Operation of large integers\file\result\_decimalSystem.txt)");

}

}

void bigInteger::subtraction\_bigInteger\_binarySystem(linkList &object1, linkList &object2) {

time\_t timeStart, timeEnd;//时间变量.

linkList result;

int objectLength1 = object1.linkListSize();

int objectLength2 = object2.linkListSize();

time(&timeStart);//设置起始时间.

if (objectLength1 > objectLength2) {//object1的长度 > object2的长度.

int subRecord = 0;//subRecord记录借位.

int sum = 0;

node \*tempPoint1 = object1.returnHeadPoint()->prev;//tempPoint1指向object1表尾节点.

node \*tempPoint2 = object2.returnHeadPoint()->prev;//tempPoint2指向object2表尾节点.

result.createLinkListWithoutForValue(objectLength1);//创建结果链表长度为object1和object2中长度较大者.

node \*resultTempPoint = result.returnHeadPoint()->prev;//resultTempPoint指向result表尾节点.

for (int i = 0; i < objectLength2; ++i) {//做object1和object2中长度较小者的长度次循环.

if ((tempPoint1->data - subRecord) < tempPoint2->data) {//需要借位.

sum = tempPoint1->data + 2 - tempPoint2->data - subRecord;//sum记录每位的差.

subRecord = 1;//subRecord记录借位.

} else {//不需要借位.

sum = tempPoint1->data - tempPoint2->data - subRecord;//sum记录每位的差.

subRecord = 0;//subRecord记录借位.

}

tempPoint1 = tempPoint1->prev;

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = sum;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

}

for (int i = objectLength2; i < objectLength1; ++i) {//做相差长度次循环.

if (tempPoint1->data < subRecord) {//若此位为0.

sum = tempPoint1->data - subRecord + 2;//从前一位借1,sum记录每位相减的差.

subRecord = 1;//sub记录借位.

} else {//若此位不为为0.

sum = tempPoint1->data - subRecord;

subRecord = 0;//sub记录借位.

}

tempPoint1 = tempPoint1->prev;

resultTempPoint->data = sum;

resultTempPoint=resultTempPoint->prev;

}

if (sum == 0) {

result.erase(1);

}

} else if (objectLength1 < objectLength2) {//object1的长度 < object2的长度.

int subRecord = 0;//subRecord记录借位.

int sum = 0;

node \*tempPoint1 = object1.returnHeadPoint()->prev;//tempPoint1指向object1表尾节点.

node \*tempPoint2 = object2.returnHeadPoint()->prev;//tempPoint2指向object2表尾节点.

result.createLinkListWithoutForValue(objectLength2);//创建结果链表长度为object1和object2中长度较大者.

node \*resultTempPoint = result.returnHeadPoint()->prev;//resultTempPoint指向result表尾节点.

for (int i = 0; i < objectLength1; ++i) {//做object1和object2中长度较小者的长度次循环.

if ((tempPoint2->data - subRecord) < tempPoint1->data) {//需要借位.

sum = tempPoint2->data + 2 - tempPoint1->data - subRecord;//sum记录每位的差.

subRecord = 1;//subRecord记录借位.

} else {//不需要借位.

sum = tempPoint2->data - tempPoint1->data - subRecord;//sum记录每位的差.

subRecord = 0;//subRecord记录借位.

}

tempPoint1 = tempPoint1->prev;

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = sum;//将结果计入和链表中.

resultTempPoint = resultTempPoint->prev;

}

for (int i = objectLength1; i < objectLength2; ++i) {//做相差长度次循环.

if (tempPoint2->data < subRecord) {//若此位为0.

sum = tempPoint2->data - subRecord + 2;//从前一位借1,sum记录每位相减的差.

subRecord = 1;//sub记录借位.

} else {//若此位不为为0.

sum = tempPoint2->data - subRecord;

subRecord = 0;//sub记录借位.

}

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = sum;

resultTempPoint=resultTempPoint->prev;

}

if (sum == 0) {

result.erase(1);

}

} else {//object1的长度 = object2的长度.

int sizeMark = 0;

node \*tempPoint1 = object1.returnHeadPoint()->prev;//tempPoint1指向object1表尾节点.

node \*tempPoint2 = object2.returnHeadPoint()->prev;//tempPoint2指向object2表尾节点.

node \*markPoint1 = object1.returnHeadPoint()->next;//tempPoint1指向object1第一个节点.

node \*markPoint2 = object2.returnHeadPoint()->next;//tempPoint2指向object2第一个节点.

//循环用于比较两个长整数大小.

for (int i = 0; i < objectLength1; ++i) {

if (markPoint1->data > markPoint2->data) {

sizeMark = 1;//第一个长整数大,m赋值为1.

break;

} else if (markPoint1->data < markPoint2->data) {

sizeMark = 2;//第二个长整数大,m赋值为2.

break;

}

markPoint1 = markPoint1->next;

markPoint2 = markPoint2->next;

}

if (sizeMark == 0) {//两个长整数相等.

result.createLinkListWithoutForValue(1);

result.returnHeadPoint()->next->data = 0;//最后计算结果为0.

}

if (sizeMark == 1) {//第一个长整数大.

result.createLinkListWithoutForValue(objectLength1);//链表长度为二者之中任意长度.

node \*resultTempPoint = result.returnHeadPoint()->prev;//resultTempPoint指向result表尾节点.

int subRecord = 0;//subRecord记录借位.

int sum = 0;

for (int i = 0; i < objectLength1; i++) {//循环位数次.

if ((tempPoint1->data - subRecord) < tempPoint2->data) {

sum = tempPoint1->data + 2 - tempPoint2->data - subRecord;//从前一位借1,sum记录每位相减的差.

subRecord = 1;//subRecord记录借位.

} else {

sum = tempPoint1->data - tempPoint2->data - subRecord;//直接相减.

subRecord = 0;//sub记录借位.

}

tempPoint1 = tempPoint1->prev;

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = sum;//将结果记入链表.

resultTempPoint = resultTempPoint->prev;

}

if (sum == 0) {//若最大数位为0.

result.erase(1);//将第一个节点,即数据0删除.

}

node \*temp1 = result.returnHeadPoint()->next;

node \*temp2 = temp1->next;

node \*temp3 = result.returnHeadPoint();

while (temp1->data == 0) {//存在0001的情况,循环用于删除多余0.

delete temp1;//删除该节点.

temp1 = temp2;

temp2 = temp2->next;

result.returnLinkListLength() = result.returnLinkListLength() - 1;//链表长度减1.

}

temp3->next = temp1;//删除节点之后的连接操作.

temp1->prev = temp3;

} else if (sizeMark == 2) {//第二个长整数大

result.createLinkListWithoutForValue(objectLength1);//链表长度为二者之中任意长度.

node \*resultTempPoint = result.returnHeadPoint()->prev;//resultTempPoint指向result表尾节点.

int subRecord = 0;//subRecord记录借位.

int sum = 0;

for (int i = 0; i < objectLength1; i++) {//循环位数次.

if ((tempPoint2->data - subRecord) < tempPoint1->data) {

sum = tempPoint2->data + 2 - tempPoint1->data - subRecord;//从前一位借1,sum记录每位相减的差.

subRecord = 1;//sub记录借位.

} else {

sum = tempPoint2->data - tempPoint1->data - subRecord;//直接相减.

subRecord = 0;//sub记录借位.

}

tempPoint1 = tempPoint1->prev;

tempPoint2 = tempPoint2->prev;

resultTempPoint->data = sum;//将结果记入链表.

resultTempPoint = resultTempPoint->prev;

}

if (sum == 0) {//若最大数位为0.

result.erase(1);//将第一个节点,即数据0删除.

}

node \*temp1 = result.returnHeadPoint()->next;

node \*temp2 = temp1->next;

node \*temp3 = result.returnHeadPoint();

while (temp1->data == 0) {//存在0001的情况,循环用于删除多余0.

delete temp1;//删除该节点.

temp1 = temp2;

temp2 = temp2->next;

result.returnLinkListLength() = result.returnLinkListLength() - 1;//链表长度减1.

}

temp3->next = temp1;//删除节点之后的连接操作.

temp1->prev = temp3;

}

}

cout << result << endl;

time(&timeEnd);

cout << "time: " << timeEnd - timeStart << endl;

cout << "Do you want to save the results in a file:(1.Yes/2.No)";

int choice;

cin >> choice;

if (choice == 1) {

string str = translateToString(result);

writeFile(str, R"(C:\Users\15590\Desktop\Operation of large integers\file\result\_binarySystem.txt)");

}

}

void bigInteger::multiplication\_bigInteger(linkList &operation1, linkList &operation2, int base) {

linkList result;

time\_t timeStart, timeEnd;//时间变量.

time(&timeStart);//设置起始时间.

if (operation1.linkListFlag() == operation2.linkListFlag()) {

result = multiplicationAuxiliaryFunction(operation1, operation2, base);

cout << result;

cout << endl;

} else {

result = multiplicationAuxiliaryFunction(operation1, operation2, base);

result.returnLinkListFlag() = -1;

cout << result;

cout << endl;

}

time(&timeEnd);

cout << "time: " << timeEnd - timeStart << endl;

cout << "Do you want to save the results in a file:(1.Yes/2.No)";

int choice;

cin >> choice;

if (choice == 1) {

string str = translateToString(result);

writeFile(str, R"(C:\Users\15590\Desktop\Operation of large integers\file\result\_decimalSystem.txt)");

}

}

void bigInteger::multiplication\_bigInteger\_binarySystem(linkList &object1, linkList &object2) {

time\_t timeStart, timeEnd;//时间变量.

linkList result;

int objectLength1 = object1.linkListSize();

int objectLength2 = object2.linkListSize();

time(&timeStart);//设置起始时间.

//judgePoint1和judgePoint2用来判断两个链表是否为0链表.

node \*judgePoint1 = object1.returnHeadPoint()->next;

node \*judgePoint2 = object2.returnHeadPoint()->next;

node \*tempPoint2 = object2.returnHeadPoint()->prev;//tempPoint2指向object2表尾节点.

//两个大整数其中有一个为0.

if (judgePoint1->data == 0 || judgePoint2->data == 0) {

result.insert(0,0);//相乘结果为0.

} else {

result.createLinkListWithForValue(objectLength1 + objectLength2);//积链表长度为两个链表长度之和,并全部初始化.

//markPoint负责把每次相乘的结果记入链表,tempPoint往前对位.

node \*tempPoint = result.returnHeadPoint()->prev;

node \*markPoint = tempPoint;

for (int i = 0; i < objectLength2; ++i) {

if (tempPoint2->data == 0) {

tempPoint2 = tempPoint2->prev;

tempPoint = tempPoint->prev;

markPoint = tempPoint;

continue;

}

int temp = 0;

node \*tempPoint1 = object1.returnHeadPoint()->prev;//tempPoint1指向object1表尾节点.

for (int j = 0; j < objectLength1; ++j) {

//每次需要相加三部分数字.

temp = temp + (tempPoint2->data) \* (tempPoint1->data) + markPoint->data;

markPoint->data = temp % 2;//结果.

temp = temp / 2;//进位.

tempPoint1 = tempPoint1->prev;

markPoint = markPoint->prev;

}

markPoint->data = markPoint->data + temp;//进位后的结果记入链表.

tempPoint2 = tempPoint2->prev;

tempPoint = tempPoint->prev;

markPoint = tempPoint;

}

if (result.returnHeadPoint()->next->data == 0)//最大数位为0.

result.erase(1);//删除0.

}

cout << result << endl;

time(&timeEnd);

cout << "time: " << timeEnd - timeStart << endl;

cout << "Do you want to save the results in a file:(1.Yes/2.No)";

int choice;

cin >> choice;

if (choice == 1) {

string str = translateToString(result);

writeFile(str, R"(C:\Users\15590\Desktop\Operation of large integers\file\result\_binarySystem.txt)");

}

}

void bigInteger::division\_bigInteger(linkList &operation1, linkList &operation2, int base) {

linkList result;

time\_t timeStart, timeEnd;//时间变量.

time(&timeStart);//设置起始时间.

if (operation1.linkListFlag() == operation2.linkListFlag()) {

result = divisionAuxiliaryFunction(operation1, operation2, base);

cout << result;

cout << endl;

} else {

result = divisionAuxiliaryFunction(operation1, operation2, base);

result.returnLinkListFlag() = -1;

cout << result;

cout << endl;

}

time(&timeEnd);

cout << "time: " << timeEnd - timeStart << endl;

cout << "Do you want to save the results in a file:(1.Yes/2.No)";

int choice;

cin >> choice;

if (choice == 1) {

string str = translateToString(result);

writeFile(str, R"(C:\Users\15590\Desktop\Operation of large integers\file\result\_decimalSystem.txt)");

}

}

void bigInteger::division\_bigInteger\_binarySystem(linkList &object1, linkList &object2) {

linkList result;

time\_t timeStart, timeEnd;//时间变量.

time(&timeStart);//设置起始时间.

if (comparisonOfSize(object1, object2) == -1) {//除数比被除数大的情况.

result.insert(0, 0);

} else {//除数和被除数相等或被除数比除数大的情况.

if (object2.returnHeadPoint()->next->data == 0) {//除数为0的情况.

cout << "Divide by 0, no result!" << endl;

} else if (object1.returnHeadPoint()->next->data == 0) {//被除数为0的情况.

result.insert(0, 0);

} else {//其余情况.

linkList tempList;

int objectLength1 = object1.linkListSize();

int objectLength2 = object2.linkListSize();

if ((objectLength2 - 1) != 0) {

tempList.createLinkListWithoutForValue(objectLength2 - 1);

node \*tempListPtr = tempList.returnHeadPoint()->next;

node \*ptr = object1.returnHeadPoint()->next;

for (int i = 0; i < objectLength2 - 1; ++i) {//循环赋值.

tempListPtr->data = ptr->data;

tempListPtr = tempListPtr->next;

ptr = ptr->next;

}

}

node \*markPtr = object1.returnHeadPoint()->next;

for (int i = 0; i < objectLength2 - 1; ++i) {//寻找object2最高位位于object的位置.

markPtr = markPtr->next;

}

node \*erasePoint;

for (int i = objectLength2 - 1; i < objectLength1; ++i) {

int count = 0;//count记录相减次数.

erasePoint = tempList.returnHeadPoint()->next;

if (erasePoint->data == 0) {//相减之后差为0,将0后的数代替0.

node \*change = tempList.returnHeadPoint()->next;

change->data = markPtr->data;

} else {//继续插入下一位数.

tempList.insert(markPtr->data, tempList.linkListSize());

}

while ((comparisonOfSize(tempList, object2)) >= 0) {//比较两个数的大小.

tempList = subtractionAuxiliaryFunction(tempList, object2, 2);//做连减操作.

count++;//记录连减次数.

}

markPtr = markPtr->next;

result.insert(count, result.linkListSize());//将每次连减的次数插入到result中.

}

if (result.returnHeadPoint()->next->data == 0) {//若最大数为0.

result.erase(1);

}

}

}

cout << result << endl;

time(&timeEnd);

cout << "time: " << timeEnd - timeStart << endl;

cout << "Do you want to save the results in a file:(1.Yes/2.No)";

int choice;

cin >> choice;

if (choice == 1) {

string str = translateToString(result);

writeFile(str, R"(C:\Users\15590\Desktop\Operation of large integers\file\result\_binarySystem.txt)");

}

}

void bigInteger::exponentialCalculation\_bigInteger(linkList &operation1, linkList &operation2, int base) {

linkList result;

time\_t timeStart, timeEnd;//时间变量.

time(&timeStart);//设置起始时间.

if (operation2.linkListSize() == 1 && operation2.returnHeadPoint()->next->data == 0) {//指数是0的情况.

result.insert(1, 0);

} else if (operation2.linkListSize() == 1 && operation2.returnHeadPoint()->next->data == 1) {//指数是1的情况.

result = operation1;

} else {

linkList tempList;

tempList.insert(1, 0);

result = multiplicationAuxiliaryFunction(operation1, operation1, base);//做一次乘法,平方.

operation2 = divisionFunction(operation2, tempList, base);//循环次数减1.

while ((comparisonOfSize(operation2, tempList)) == 1) {

result = multiplicationAuxiliaryFunction(result, operation1, base);//连乘.

operation2 = subtractionAuxiliaryFunction(operation2, tempList, 10);//循环次数减1.

}

}

cout << result << endl;

time(&timeEnd);

cout << "time: " << timeEnd - timeStart << endl;

cout << "Do you want to save the results in a file:(1.Yes/2.No)";

int choice;

cin >> choice;

if (choice == 1) {

string url;

if (base == 10) {

url = R"(C:\Users\15590\Desktop\Operation of large integers\file\result\_decimalSystem.txt)";

} else {

url = R"(C:\Users\15590\Desktop\Operation of large integers\file\result\_binarySystem.txt)";

}

string str = translateToString(result);

writeFile(str, url);

}

}

===============================================================================

Action.h

#ifndef OPERATION\_OF\_LARGE\_INTEGERS\_ACTION\_H

#define OPERATION\_OF\_LARGE\_INTEGERS\_ACTION\_H

#include <iostream>

#include "LinkList.h"

#include "BigInteger.h"

using namespace std;

//操作类.

class action {

public:

static void welcomeScreen();

static void directorySelectionScreen();

static void baseSelection();

static void exitSystem();

static void selectionDirectory();

static void operationWithoutFile();

static void operationWithFile();

};

#endif //OPERATION\_OF\_LARGE\_INTEGERS\_ACTION\_H

===============================================================================

Action.cpp

#include "Action.h"

/\*\*

\* @details 欢迎界面.

\* @param void.

\* @return void.

\*/

void action::welcomeScreen() {

cout << "\*\*\*长整数运算程序\*\*\*\n";

cout << "!该程序根据屏幕提示选择想要进行的操作\n\n";

cout << "作者：郑东晖\n";

cout << "班级：大数据工程应用1901班\n";

cout << "学号：201906062631\n";

cout << "\n";

}

/\*\*

\* @details 目录选择界面界面.

\* @param void.

\* @return void.

\*/

void action::directorySelectionScreen() {

cout << "请选择：\n";

cout << "1.不涉及文件的大整数运算\n";

cout << "2.涉及文件的大整数运算\n";

cout << "3.退出程序\n";

cout << "你的选择：";

}

/\*\*

\* @details 选择进制运算.

\* @param void.

\* @return void.

\*/

void action::baseSelection() {

cout << "请选择:\n";

cout << "1.十进制\n";

cout << "2.二进制\n";

cout << "你的选择：";

}

/\*\*

\* @details 退出程序.

\* @param void.

\* @return void.

\*/

void action::exitSystem() {

cout << "\n欢迎下次使用！\n\n";

system("pause");

exit(0);

}

/\*\*

\* @details 运算方法.

\* @param void.

\* @return void.

\*/

void action::selectionDirectory() {

cout << "输入1、2、3、4或5选择操作：\n";

cout << "1、加法\n";

cout << "2、减法\n";

cout << "3、乘法\n";

cout << "4、除法\n";

cout << "5、指数\n";

cout << "6、退出\n";

cout << "你的选择：";

}

/\*\*

\* @details 不涉及文件的大整数计算.

\* @param void.

\* @return void.

\*/

void action::operationWithoutFile() {

int choice;

baseSelection();//调用进制选择函数.

cin >> choice;

fflush(stdin);//清除缓存.

while (choice > 2 || choice < 1) {

cout << "您的输入有误，请重新输入！\n" << endl;

baseSelection();//调用进制选择函数.

cin >> choice;

fflush(stdin);//清除缓存.

}

if (choice == 1) {//十进制.

int flag = 1;

while (flag) {

int operatorSelection;

selectionDirectory();//调用运算方法函数.

cin >> operatorSelection;

fflush(stdin);//清除缓存.

while (operatorSelection > 6 || operatorSelection < 1) {

cout << "您的输入有误，请重新输入！\n" << endl;

selectionDirectory();//调用运算方法函数.

cin >> operatorSelection;

fflush(stdin);//清除缓存.

}

linkList list1, list2;

switch (operatorSelection) {

case 1: {//加法.

bigInteger::inputListWithoutFile(list1, list2);

cout << "list1:" << list1 << endl;

cout << "list2:" << list2 << endl;

cout << "加法 —>>> (list1 + list2) = ";

bigInteger::addition\_bigInteger(list1, list2, 10);

break;

}

case 2: {//减法.

bigInteger::inputListWithoutFile(list1, list2);

cout << "list1:" << list1 << endl;

cout << "list2:" << list2 << endl;

cout << "减法 —>>> (list1 - list2) = ";

bigInteger::subtraction\_bigInteger(list1, list2, 10);

break;

}

case 3: {//乘法.

bigInteger::inputListWithoutFile(list1, list2);

cout << "list1:" << list1 << endl;

cout << "list2:" << list2 << endl;

cout << "乘法 —>>> (list1 \* list2) = ";

bigInteger::multiplication\_bigInteger(list1, list2, 10);

break;

}

case 4: {//除法.

bigInteger::inputListWithoutFile(list1, list2);

cout << "list1:" << list1 << endl;

cout << "list2:" << list2 << endl;

cout << "除法 —>>> (list1 / list2) = ";

bigInteger::division\_bigInteger(list1, list2, 10);

break;

}

case 5: {//指数运算.

bigInteger::inputListWithoutFile(list1, list2);

cout << "list1:" << list1 << endl;

cout << "list2:" << list2 << endl;

cout << "指数运算 —>>> (list1 ^ list2) = ";

bigInteger::exponentialCalculation\_bigInteger(list1, list2, 10);

break;

}

default: {

break;

}

}

if (operatorSelection == 6) {

cout << "Exit?(0.Yes/1.No):";

cin >> choice;

flag = choice;

}

}

} else {//二进制.

int flag = 1;

while (flag) {

int operatorSelection;

selectionDirectory();//调用运算方法函数.

cin >> operatorSelection;

fflush(stdin);//清除缓存.

while (operatorSelection > 6 || operatorSelection < 1) {

cout << "您的输入有误，请重新输入！\n" << endl;

selectionDirectory();//调用运算方法函数.

cin >> operatorSelection;

fflush(stdin);//清除缓存.

}

linkList list1, list2;

switch (operatorSelection) {

case 1: {//加法.

bigInteger::inputListWithoutFile(list1, list2);

cout << "list1:" << list1 << endl;

cout << "list2:" << list2 << endl;

cout << "加法 —>>> (list1 + list2) = ";

bigInteger::addition\_bigInteger\_binarySystem(list1, list2);

break;

}

case 2: {//减法.

bigInteger::inputListWithoutFile(list1, list2);

cout << "list1:" << list1 << endl;

cout << "list2:" << list2 << endl;

cout << "减法 —>>> (list1 - list2) = ";

bigInteger::subtraction\_bigInteger\_binarySystem(list1, list2);

break;

}

case 3: {//乘法.

bigInteger::inputListWithoutFile(list1, list2);

cout << "list1:" << list1 << endl;

cout << "list2:" << list2 << endl;

cout << "乘法 —>>> (list1 \* list2) = ";

bigInteger::multiplication\_bigInteger\_binarySystem(list1, list2);

break;

}

case 4: {//除法.

bigInteger::inputListWithoutFile(list1, list2);

cout << "list1:" << list1 << endl;

cout << "list2:" << list2 << endl;

cout << "除法 —>>> (list1 / list2) = ";

bigInteger::division\_bigInteger\_binarySystem(list1, list2);

break;

}

case 5: {//指数运算.

bigInteger::inputListWithoutFile(list1, list2);

cout << "list1:" << list1 << endl;

cout << "list2:" << list2 << endl;

cout << "指数运算 —>>> (list1 ^ list2) = ";

bigInteger::exponentialCalculation\_bigInteger(list1, list2, 2);

break;

}

default: {

break;

}

}

if (operatorSelection == 6) {

cout << "Exit?(0.Yes/1.No):";

cin >> choice;

flag = choice;

}

}

}

}

/\*\*

\* @details 涉及文件的大整数计算.

\* @param void.

\* @return void.

\*/

void action::operationWithFile() {

int choice;

baseSelection();//调用进制选择函数.

cin >> choice;

fflush(stdin);//清除缓存.

while (choice > 2 || choice < 1) {

cout << "您的输入有误，请重新输入！\n" << endl;

baseSelection();//调用进制选择函数.

cin >> choice;

fflush(stdin);//清除缓存.

}

if (choice == 1) {//十进制.

int flag = 1;

while (flag) {

int operatorSelection;

selectionDirectory();//调用运算方法函数.

cin >> operatorSelection;

fflush(stdin);//清除缓存.

while (operatorSelection > 6 || operatorSelection < 1) {

cout << "您的输入有误，请重新输入！\n" << endl;

selectionDirectory();//调用运算方法函数.

cin >> operatorSelection;

fflush(stdin);//清除缓存.

}

linkList list1, list2;

switch (operatorSelection) {

case 1: {//加法.

bigInteger::listOperateWithFile\_decimalSystem(list1, list2, '+');

break;

}

case 2: {//减法.

bigInteger::listOperateWithFile\_decimalSystem(list1, list2, '-');

break;

}

case 3: {//乘法.

bigInteger::listOperateWithFile\_decimalSystem(list1, list2, '\*');

break;

}

case 4: {//除法.

bigInteger::listOperateWithFile\_decimalSystem(list1, list2, '/');

break;

}

case 5: {//指数运算.

bigInteger::listOperateWithFile\_decimalSystem(list1, list2, '^');

break;

}

default: {

break;

}

}

if (operatorSelection == 6) {

cout << "Exit?(0.Yes/1.No):";

cin >> choice;

flag = choice;

}

}

} else {//二进制.

int flag = 1;

while (flag) {

int operatorSelection;

selectionDirectory();//调用运算方法函数.

cin >> operatorSelection;

fflush(stdin);//清除缓存.

while (operatorSelection > 6 || operatorSelection < 1) {

cout << "您的输入有误，请重新输入！\n" << endl;

selectionDirectory();//调用运算方法函数.

cin >> operatorSelection;

fflush(stdin);//清除缓存.

}

linkList list1, list2;

switch (operatorSelection) {

case 1: {//加法.

bigInteger::listOperateWithFile\_binarySystem(list1, list2, '+');

break;

}

case 2: {//减法.

bigInteger::listOperateWithFile\_binarySystem(list1, list2, '-');

break;

}

case 3: {//乘法.

bigInteger::listOperateWithFile\_binarySystem(list1, list2, '\*');

break;

}

case 4: {//除法.

bigInteger::listOperateWithFile\_binarySystem(list1, list2, '/');

break;

}

case 5: {//指数运算.

bigInteger::listOperateWithFile\_binarySystem(list1, list2, '^');

break;

}

default: {

break;

}

}

if (operatorSelection == 6) {

cout << "Exit?(0.Yes/1.No):";

cin >> choice;

flag = choice;

}

}

}

}

===============================================================================

main.cpp

#include <iostream>

#include "Action.h"

using namespace std;

int main() {

action::welcomeScreen();//调用欢迎界面.

system("pause");

system("cls");

while (true) {//死循环.

int choice;//用户输入的选择.

int flag = 1;//用于判断是否退出运算.

action::directorySelectionScreen();//调用目录选择界面.

cin >> choice;

fflush(stdin);//清除缓存.

while (choice > 3 || choice < 1) {

cout << "您的输入有误，请重新输入！\n" << endl;

action::directorySelectionScreen();//调用目录选择界面.

cin >> choice;

fflush(stdin);//清除缓存.

}

int continueKey;

switch (choice) {

case 1://不涉及文件的大整数计算.

while (flag) {//运算操作.

action::operationWithoutFile();//调用operationWithoutFile函数.

cout << "\n继续运算？（选择1：是/0：否）：";

fflush(stdin);//清除缓存.

cin >> continueKey;

flag = continueKey;

system("cls");

}

break;

case 2://涉及文件的大整数计算.

while (flag) {//运算操作.

action::operationWithFile();//调用operationWithFile函数.

cout << "\n继续运算？（选择1：是/0：否）：";

fflush(stdin);//清除缓存.

cin >> continueKey;

flag = continueKey;

system("cls");

}

break;

default:

action::exitSystem();

break;

}

}

return 0;

}