云南大学数学与与统计学院

上机实践报告

课程名称：数据结构与算法实验	**年级**：2013	**考试成绩**：
指导教师：陆正福	**姓名**：金洋	
上机实践名称：**DS&A 期中考试实验**	**学号**：20131910023	
分组成员（学号-姓名）：20131910023-金洋 2013190069-王涵 **组号**：4		

一、实验目的

应用所学数据结构与算法的基本知识和技能，完成从基本数据类型到抽象数据类型、复杂数据结构和算法的设计。分组完成实验，题目分配参见题号分配表。

二、实验内容

第 1 题：

（1）以整型数（int）为基础，设计任意长度整数的大整数运算系统（记为 MyBigInt）。

（2）以上述的 MyBigInt 型整数运算系统为基础，设计以 MyBigInt 型整数为系数的多项式运算系统（记为 MyPolyBigInt）。

（3）以 MyPolyBigInt 为基础，设计有理分式运算系统(记为 MyRatPolyBigInt)。

要求完成:

A. ADT 设计；
B. 选择合适的存储结构；
C. 关键算法的设计；
D. 基于 Java interface 的 ADT 表示 （上机完成）；
E. 基于 Java class 的具体实现（上机完成）。

三、实验平台

个人计算机; Oracle/Sun Java 7 SE 或 EE

四、实验记录与实验结果分析

（注意记录实验中遇到的问题。实验报告的评分依据之一是实验记录的细致程度、实验过程的真实性、实验结果的解释和分析。**如果涉及实验结果截屏，应选择白底黑字。**）

1.以整型数（int）为基础，设计任意长度整数的大整数运算系统（记为 MyBigInt）。

本题要求设计任意长度的整数运算系统，对于 java 中的 int，取值为-2147483638~2147483637，长度 10 位；对于 long，取值为-9223372036854775808~9223372036854775807，长度 19 位；倘若需要计算的数的长度超过 19 位，想要得到精确值，java 本身存有的基本数据类型不再适合。

因此，我们需要构造一个新的数据结构，使其满足任意长度的整数的存储要求。

为此定义一个 BigNumStruct 类。

存储大数有链式和顺序存储方式两种方法。链式可以适应不定长度的大整数，但由于存储空间是离散的，故随机访问效率不高；顺序存储主要是数组方式，也是 BigNumStruct 类采用的存储大数的结构，其存储空间是连续的，随机访问率高，同时为了使数组适应不定长度的大整数，BigNumStruct 开辟的数组长度并不是固定的，而是根据输入的参与运算的参数的长度，再开辟相应长度的数组来存储大数。

BigNumStruct 包括如下属性：

```
protected static final int STEP=9,MAX_INT=1000000000;//STEP 为每个数组单元中能存的最大位数，MAX_INT 为每个数组单元中能存的最大数
protected StringBuffer strBI;//字符串型大整数，含符号;由于要处理符号和开头的 0 所以设为可变字符串型
protected byte sign;//大整数的正负性，正数和零时 sign=' ';负数时，sign='-';
protected int size;//无符号大整数的长度
protected int arrayLen;//记录实际数组长度，因为用 Arrays。length 得到的可能会比实际存储了数的数组长度多一位，所以必须再设一个变量 arrayLen;
protected int[] array;//存储大整数，以数组形式存储，不含符号位
```

BigNumStruct 类含 3 中构造方法，一种为无参构造方法；一种是参数为输入的字符串的构造方法，用于参与运算的参数的存储构造；一种是参数为大整数长度的构造方法，用于为结果开辟存储空间；

BigNumStruct 该类将一个大整数的分段存储在一个 int 型数组中，为了充分利用空间，采用了 1，000，000，000 进制，即每九位为一个单元。按照笔算的习惯，我们将大数按照从低位到高位依次存储在数组中；

有了 **BigNumStruct** 结构来存储大数，就可以来实现 **MyBigInt**，其主要要实现的功能为加减乘数四个运算，各运算基本思路如下：

（1）加法 **add**：

加法可分为，符号相同的两数、符号不同的两数的加法:

符号相同的两数，长度相同部分相加，另一数的剩余长度直接放到结果中；

符号不同的两数加法，亦可以说是减法；由于不知道哪个加数大，做完减法可能会出现高位是负数的问题；为了减少对可能结果的分类讨论，我们先将两个加数的无符号位进行比较，大的作为被减数，小的作为减数，这样做完减法不会出现高位为负数的问题；而结果的符号和被减数的符号一致；

（2）减法 minus：

减法只需将减数的符号乘-1，再调用加法即可

(3)乘法 multiply：

同笔算乘法，bigInt2 从低位开始遍乘 bigInt1 的每一位，bigInt1[i]与 bigInt2[j]的结果累加在 ansBigInt[i+j]中；

(4)除法 devide：

除法比较复杂，一种思路是被除数连续减除数，差作为新的被除数，直到被减数<减数；但当两数的长度相差很大时，其时间效率很低；

另一个思路先估商，采用二分法试商，下界为 1，上界为被除数，每次求出上下界平均数 mid:

mid*除数>被除数：上界=mid-1;

mid*除数<被除数：下界=mid+1;

mid*除数=被除数：商=mid，余数=0;

（5）带小数的除法：

带小数的除法，商分为整数位和小数点位，在调用完 devide 后，商和余数均已求出，整数位即为商;

对于小数位，只需将余数的末尾补 0 至与除数一样长度，保留 k 位小数，则继续在末尾添 k 个 0，将添完 0 后的余数作为新的被除数，除数不变，再调用 devide，结果的商即为小数位；（考虑到本实验要求设计整数运算系统，本实验报告并未将带小数的除法基于 Java class 的具体实现代码附上）

基于 Java interface 的 ADT 表示和 基于 Java class 的具体实现如下：

IMyBigInt.java

```
package Midterm;
public interface IMyBigInt {

	//比较两个无符号大整数的大小
	public byte compare(BigNumStruct bigInt1,BigNumStruct bigInt2);

	//加法计算
	public BigNumStruct add(BigNumStruct bigInt1,BigNumStruct bigInt2);

	//减法计算
	public BigNumStruct minus(BigNumStruct bigInt1,BigNumStruct bigInt2);

	//乘法计算
	public BigNumStruct multiply(BigNumStruct bigInt1,BigNumStruct bigInt2);

	//除法计算，产生商和余数
	public void devide(BigNumStruct bigInt1,BigNumStruct bigInt2)
			throws ArithmeticException;

	//求两数中值
	public BigNumStruct half(BigNumStruct a,BigNumStruct b);

	//求出大数的数组的实际长度
	public void changArrayLen(BigNumStruct ans);

	//将结果转化为字符串
	public String getAns(BigNumStruct ansBigInt);

	//调用除法过后，再用 getQuoBigInt()可得到商;
	public BigNumStruct getQuoBigInt();
```

```
	//调用除法过后，再用 getRemBigInt()可得到余数；
	public BigNumStruct getRemBigInt();
}
```

BigNumStruct.java

```
package Midterm;
import java.util.Arrays;
/*
 * Java 中 ，int 范围为-2147483648~2147483648，为了充分利用空间，本算法采用 1，000，000，000 进制；输入的运算参数的存储使用 int 型数组；
 */
public class BigNumStruct implements Cloneable{
	protected static final int STEP=9,MAX_INT=1000000000;//STEP 为每个数组单元中能存的最大位数，MAX_INT 为每个数组单元中能存的最大数
	protected StringBuffer strBI;//字符串型大整数，含符号;由于要处理符号和开头的 0 所以设为可变字符串型
	protected byte sign;//大整数的正负性，正数和零时 sign=' ';负数时，sign='-';
	protected int size;//无符号大整数的长度
	protected int arrayLen;//记录实际数组长度，因为用 Arrays。length 得到的可能会比实际存储了数的数组长度多一位，所以必须再设一个变量 arrayLen；
	protected int[] array;//存储大整数，以数组形式存储，不含符号位


	//无参构造方法，不在此处对开辟指定长度的数组
	public BigNumStruct(){
		strBI=new StringBuffer();
		sign=1;
		size=1;
		arrayLen=1;
		//cloneFlag=0;
	}

	//有参构造方法，输入时为字符串，根据字符串来构造
	public BigNumStruct(StringBuffer s){
		strBI=s;
		if (strBI.indexOf("-")==0){
			sign=-1;
			strBI.replace(0,1,"");//将负号拿掉，使得字符串变为无符号整数
		}
		else sign=1;
		//清除开头的无效 0
		while ((strBI.indexOf("0")==0)&&(strBI.length()>1) )
			strBI.replace(0,1,"");

		size=strBI.length();

		//根据输入的大整数长度，开辟指定长度的数组
		array=new int[(size-1)/STEP+1];//例如数字 9876543210，需用两个 int 单
```

元；故开辟的长度为(size-1)/STEP+1

```
            //stringToArray
            int i=0,l,r;//i 指向数组；
            r=size;
            l=r-STEP>0?r-STEP:0;
            while (r>0){
                  //substring()取出下标为 l≤index<r 之间的子串；parseInt()使得数字字符串转为 int 型
                  array[i]=Integer.parseInt(strBI.substring(l,r));
                  r=r-STEP;
                  l=r-STEP>0?r-STEP:0;
                  i++;
            }

            //记录数组长度
            arrayLen=i;

      }

      //有参构造方法，开辟指定长度的数组，一般为存储结果来开辟
      public BigNumStruct(int length){
            strBI=new StringBuffer("");
            array=new int[length];
            arrayLen=length;
            sign=1;
            size=1;
            Arrays.fill(array, 0);
            //cloneFlag=0;
      }

      public StringBuffer getStrBI(){
            return strBI;
      }
      public byte getSign(){
            return sign;
      }
      public int size(){
            return size;
      }
      public int getArrayLen(){
            return arrayLen;
      }
      public int getArray(int i){
            return array[i];
      }
      public int[] getArray(){
            return array;
      }


      public void appendStrBI(int newInt){
            strBI.append(newInt);
```

```
	}
	public void setSign(byte newSign){
		sign=newSign;
	}
	public void setSize(int newSize){
		size=newSize;
	}
	public void setArrayLen(int newLen){
		arrayLen=newLen;
	}
	public void setArray(int i,int newEle){
		array[i]=newEle;
	}
	/*public void setCloneFlag(byte newCF){
		cloneFlag=newCF;
	}*/

	//用来比较 StringBuffer 与 String 是否相等（如判断输入的是否为 0），因为
StringBuffer 类未重写 equals()方法，比较的是地址是否相等；故此处重写 StringBuffer 类的
equals()方法
	public boolean equals(String s2){
		String s1=new String(strBI);
		return s1.equals(s2);
	}


	//大数运算结果存储在数组中，arrayToString()可以将数组中的数提取出来转化为字符串
	public String arrayToString(){
		strBI.replace(0,size,"");
		if (getSign()==-1) strBI.append('-');
		int i=arrayLen-1;
		strBI.append(array[i]);//首位数直接复制到字符串

		i--;
		//十亿进制，不足位需要补零；
		for (;i>=0;i--){
			if (getArray(i)<100000000) strBI.append(0);
			if (getArray(i)<10000000) strBI.append(0);
			if (getArray(i)<1000000) strBI.append(0);
			if (getArray(i)<100000) strBI.append(0);
			if (getArray(i)<10000) strBI.append(0);
			if (getArray(i)<1000) strBI.append(0);
			if (getArray(i)<100) strBI.append(0);
			if (getArray(i)<10) strBI.append(0);
			appendStrBI(getArray(i));
		}
		size=strBI.length();
		return strBI.toString();


	}
	//克隆，为了不改变原数
	public BigNumStruct clone(){
```

```
            BigNumStruct bI = null;
            try {
                  bI=(BigNumStruct)super.clone();
            }
            catch (CloneNotSupportedException e) {
                  e.printStackTrace();
            }

          return bI;
      }


}
```

MyBigInt 的实现：

MyBigInt.java

```
package Midterm;
public class MyBigInt implements IMyBigInt {
      protected static final int MAX_INT=1000000000;//MAX_INT 为每个数组单元中能存的最大数
      protected BigNumStruct bigInt1,bigInt2;//为两个参数
      protected BigNumStruct quoBigInt,remBigInt;//由于除法有两个返回值：商，余数，故除法方法不能为函数方法，将两个结果设为全局变量

      public MyBigInt(){}/**/

      //有些大数计算只有一个参数，如阶乘
      public MyBigInt(StringBuffer para1){
            bigInt1=new BigNumStruct(para1);
      }

      //大数计算的构造方法，二元参数，如加减乘除
      public MyBigInt(StringBuffer para1,StringBuffer para2){
            bigInt1=new BigNumStruct(para1);
            bigInt2=new BigNumStruct(para2);
      }

      public BigNumStruct getBigInt1(){
            return bigInt1;
      }
      public BigNumStruct getBigInt2(){
            return bigInt2;
      }
      public BigNumStruct getQuoBigInt(){
            return quoBigInt;
```

```
	}
	public BigNumStruct getRemBigInt(){
		return remBigInt;
	}


	//比较两个无符号大整数的大小
	//bigInt1>bigInt2 返回 1；bigInt1<bigInt2 返回-1；bigInt1=bigInt2 返回 0；
	public byte compare(BigNumStruct bigInt1,BigNumStruct bigInt2){
		if (bigInt1.getArrayLen()>bigInt2.getArrayLen()) return 1;
		if (bigInt1.getArrayLen()<bigInt2.getArrayLen()) return -1;
		for (int i=bigInt1.getArrayLen()-1; i>=0; i--){
			if (bigInt1.getArray(i)>bigInt2.getArray(i)) return 1;
			if (bigInt1.getArray(i)<bigInt2.getArray(i)) return -1;
		}
		return 0;
	}

	//加法分为符号相同的加法；符号不同的加法
	public BigNumStruct add(BigNumStruct bigInt1,BigNumStruct bigInt2) {
		BigNumStruct ansBigInt=new BigNumStruct();

		//如果 bigInt1<bigInt2,则交换(引用的交换，时间复杂度仅为 O(1)),
		//保证无符号位 bigInt1>bigInt2,可以减少代码的书写量，作用在后面会介绍
		Byte ansCompare;
		ansCompare=compare(bigInt1,bigInt2);
		if (ansCompare<0) {BigNumStruct bigIntTemp =
bigInt1;bigInt1=bigInt2;bigInt2=bigIntTemp;}

		//如果其中一数为 0，则结果等于另一数，返回
		if (bigInt2.equals("0")){
			ansBigInt=bigInt1.clone();
			//ansBigInt.setCloneFlag((byte)(1));
			return ansBigInt;
		}


		//符号相同的加法,如  a+b，(-a)+(-b)，  (a≥0，b≥0)；两数的相同长度进行相
加，较大数的剩余长度直接放入结果里
		if (bigInt1.getSign()==bigInt2.getSign()){

			//存储结果的数据结构
			ansBigInt=new BigNumStruct(bigInt1.getArrayLen()+1);
			ansBigInt.setSign(bigInt1.getSign());

			//由于已经保证了无符号位 bigInt1>bigInt2；先进行长度较短的 bigInt2
一段长度的相加
			int carry=0,tempSum=0,i;//carry 记录进位，tempSum 为每一个数组单元
的临时和，i 为指向数组的下标
			for(i=0;i<bigInt2.getArrayLen();i++){
				tempSum=bigInt1.getArray(i)+bigInt2.getArray(i)+carry;
				ansBigInt.setArray(i,tempSum % MAX_INT);
```

```
                carry=tempSum /MAX_INT;
            }
            //处理 bigInt1 剩余长度的计算
            for (;i<bigInt1.getArrayLen();i++){
                tempSum=bigInt1.getArray(i)+carry;
                ansBigInt.setArray(i,tempSum % MAX_INT);
                carry=tempSum /MAX_INT;
            }
            if (carry!=0){
                ansBigInt.setArray(i,carry);
                i++;
            }
            //记录实际数组长度
            ansBigInt.setArrayLen(i);
            return ansBigInt;
        }

        //符号不同的加法,如 a+(-b), (-a)+b, (a>0，b>0)
        else{
            //符号不同的两数的加法，亦可以说是减法；
            //由于不知道哪个加数大，做完减法可能会出现高位是负数的问题；为了减少对可能结果的分类讨论，我们已经将两个加数的无符号位进行比较，大的作为被减数，小的作为减数，这样做完减法不会出现高位为负数的问题；



            //两者相等，则结果为 0，直接返回
            if (ansCompare==0){
                ansBigInt=new BigNumStruct(1);
                return ansBigInt;
            }

            //存储结果的数据结构
            ansBigInt=new BigNumStruct(bigInt1.getArrayLen());
            ansBigInt.setSign(bigInt1.getSign());//结果的符号一定与被减数的符号相同

            //先进行 bigInt2 一段长度的相减
            int borrow=0,tempResult=0,i;//borrow 记录借位，tempResult 为每一个数组单元的临时差，i 为指向数组的下标
            for(i=0;i<bigInt2.getArrayLen();i++){
                tempResult=bigInt1.getArray(i)-bigInt2.getArray(i)-borrow;
                if (tempResult<0) {
                    borrow=1;//借位
                    tempResult+=MAX_INT;
                }
                else borrow=0;
                ansBigInt.setArray(i, tempResult);
            }
            //处理 bigInt1 剩余长度的计算
```

```
                for (;i<bigInt1.getArrayLen();i++){
                    tempResult=bigInt1.getArray(i)-borrow;
                    if (tempResult<0) {
                        borrow=1;//借位
                        tempResult+=MAX_INT;
                    }
                    else borrow=0;
                    ansBigInt.setArray(i, tempResult);
                }

                this.changArrayLen(ansBigInt);
                return ansBigInt;
            }
        }

        //减法，只需改动减数的符号，再按照加法进行
        public BigNumStruct minus(BigNumStruct bigInt1,BigNumStruct bigInt2){

            bigInt2.setSign((byte) -bigInt2.getSign());
            return(this.add(bigInt1,bigInt2));
        }

        //乘法，如同笔算乘法；
        //两重循环，bigInt1[i]与 bigInt2[j]的结果累加在 ansBigInt[i+j]中
        public BigNumStruct multiply(BigNumStruct bigInt1,BigNumStruct bigInt2){
            BigNumStruct ansBigInt=new BigNumStruct();

            ansBigInt=new
BigNumStruct(bigInt1.getArrayLen()+bigInt2.getArrayLen());//积的长度最大为两个因数
的长度和

            //有一个为 0，则结果为 0
            if (bigInt1.equals("0")){
                ansBigInt=bigInt1.clone();
                //ansBigInt.setCloneFlag((byte)(1));
                return ansBigInt;
            }
            else if (bigInt2.equals("0")){
                ansBigInt=bigInt2.clone();
                //ansBigInt.setCloneFlag((byte)(1));
                return ansBigInt;
            }

            ansBigInt.setSign((byte) (bigInt1.getSign()*bigInt2.getSign()));//结
果的符号；两个 byte 型相乘为结果为 int，需强制转换

            //两数相乘+进位+已有积 ，这个结果是否需要 long 型临时变量？考虑极端情况：两
数相乘+进位+已有积=(10^9-1)*(10^9-1)+(10^9-1)+(10^9-1)=10^18-1>Integer.MAX_VALUE,
∴需要
            long carry=0,tempResult=0;
            int i,j;//i,j 为两个因数的数组的下标
            for (i=0;i<bigInt1.getArrayLen();i++){
```

```
                carry=0;
                for (j=0;j<bigInt2.getArrayLen();j++){

    tempResult=(long)(bigInt1.getArray(i))*(long)(bigInt2.getArray(j))+carry+(
long)(ansBigInt.getArray(i+j));//两数相乘+进位+已有积,必须强制转换
                    carry=tempResult /MAX_INT;
                    ansBigInt.setArray(i+j,(int) (tempResult % MAX_INT));

                }
                if (carry!=0)
                    ansBigInt.setArray(i+j,(int) (carry));
            }

            this.changArrayLen(ansBigInt);
            return ansBigInt;
      }


      //除法
      public void devide(BigNumStruct bigInt1,BigNumStruct bigInt2) throws
ArithmeticException{


            if (bigInt2.equals("0")) throw new ArithmeticException("除数为 0，请严
谨一点好么？ ");
            StringBuffer ze=new StringBuffer("0");
            BigNumStruct zero=new BigNumStruct(ze);//BigNumStruct 的零;

            //被除数为 0，则商为 0，余数等于除数
            if (bigInt1.equals("0")){

                quoBigInt=zero.clone();
                remBigInt=bigInt2.clone();
                //remBigInt.setCloneFlag((byte)(1));
                return;
            }

            //除数为±1，商等于被除数，余数为 0;
            if (bigInt2.equals("1")){

                quoBigInt=bigInt1.clone();
                quoBigInt.setSign((byte)
(bigInt1.getSign()*bigInt2.getSign()));
                remBigInt=zero.clone();
                return;
            }


            quoBigInt=new BigNumStruct(bigInt1.getArrayLen()-
bigInt2.getArrayLen()+1);//商的长度
            quoBigInt.setSign((byte) (bigInt1.getSign()*bigInt2.getSign()));//两
数同号，商为正，否则为负
            remBigInt=new BigNumStruct(bigInt2.getArrayLen());//余数的长度总≤除数
```

```
            remBigInt.setSign((byte) (bigInt1.getSign()));//余数的符号和被除数相同

            //二分法找出商
            //由于余数和商的符号已经确定，剩下的除法为无符号运算
            bigInt1.setSign((byte) 1);
            bigInt2.setSign((byte) 1);

            StringBuffer oneS=new StringBuffer("1");
            BigNumStruct one=new BigNumStruct(oneS);
            BigNumStruct low=one.clone();//下界为 1
            BigNumStruct high=bigInt1.clone();//上界为被除数
            BigNumStruct mid=new BigNumStruct();
            BigNumStruct tempBigInt1=new BigNumStruct(bigInt1.getArrayLen());//
用于与被除数比较


            do{
                  mid=half(low,high).clone();//mid 为此时求出的商
                  tempBigInt1=multiply(mid,bigInt2).clone();

                  int ansCompare=compare(tempBigInt1,bigInt1);
                  if (ansCompare==0){
                        quoBigInt.array=mid.getArray().clone();/**/

                        remBigInt=zero.clone();
                        return;
                  }
                  else if (ansCompare<0)
                        low=this.add(mid,one.clone());//由于参数传递的是地址，后期
减法操作会改变 one（单位 1 的大数形式）的符号值，为了使 one 不变，使用 clone()方法；
                  else
                        high=this.minus(mid,one.clone());
            }while (compare(low,high)<=0);

            tempBigInt1=multiply(mid,bigInt2);//mid 为经过二分法，此时最接近商的数
            //试得商正合适
            if (compare(tempBigInt1,bigInt1)<=0){
                  quoBigInt=mid.clone();
                  remBigInt=minus(bigInt1,tempBigInt1.clone());
            }
            //试得的商过大，需减一
            else{
                  quoBigInt=minus(mid,one.clone());
                  remBigInt=minus(bigInt1,minus(tempBigInt1,bigInt2.clone()));//
‘过大的临时被除数’-除数=真实商*除数，
            }

            this.changArrayLen(quoBigInt);
            this.changArrayLen(remBigInt);
      }

      //求中值
```

```
	public BigNumStruct half(BigNumStruct a,BigNumStruct b){
		BigNumStruct temp=add(a,b);
		int i=temp.getArrayLen()-1,t=0;
		for (;i>0;i--){
			t=temp.getArray(i);
			temp.setArray(i, t/2);
			if (t%2==1) temp.setArray(i-1, temp.getArray(i-1)+MAX_INT);
		}
		temp.setArray(0,temp.getArray(0)/2);

		//实际长度
		this.changArrayLen(temp);
		return temp;
	}

	//求出大数的数组的实际长度
	public void changArrayLen(BigNumStruct ans){
		int i;
		i=ans.getArrayLen()-1;
		while(ans.getArray(i)==0 && i>0) i--;
		ans.setArrayLen(i+1);
	}

	//将结果转化为字符串
	public String getAns(BigNumStruct ansBigInt){
		return ansBigInt.arrayToString();
	}

}
```

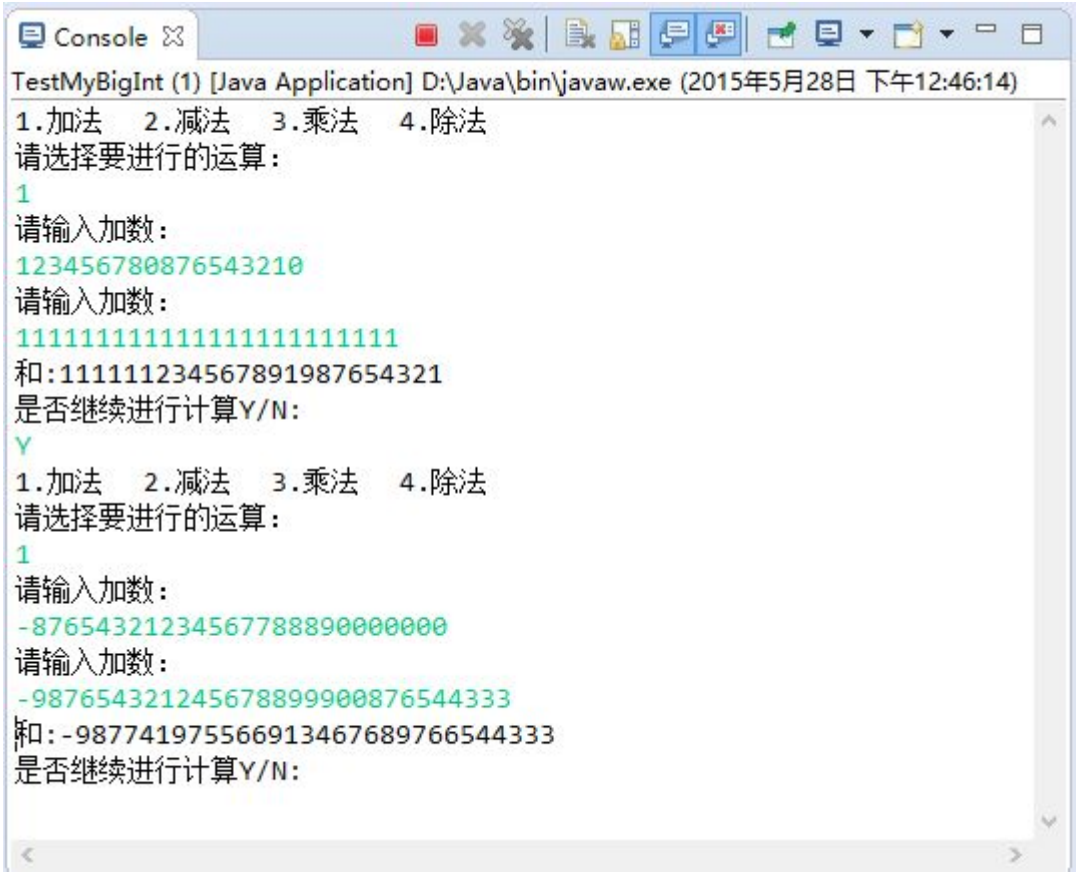
测试 MyBigInt：

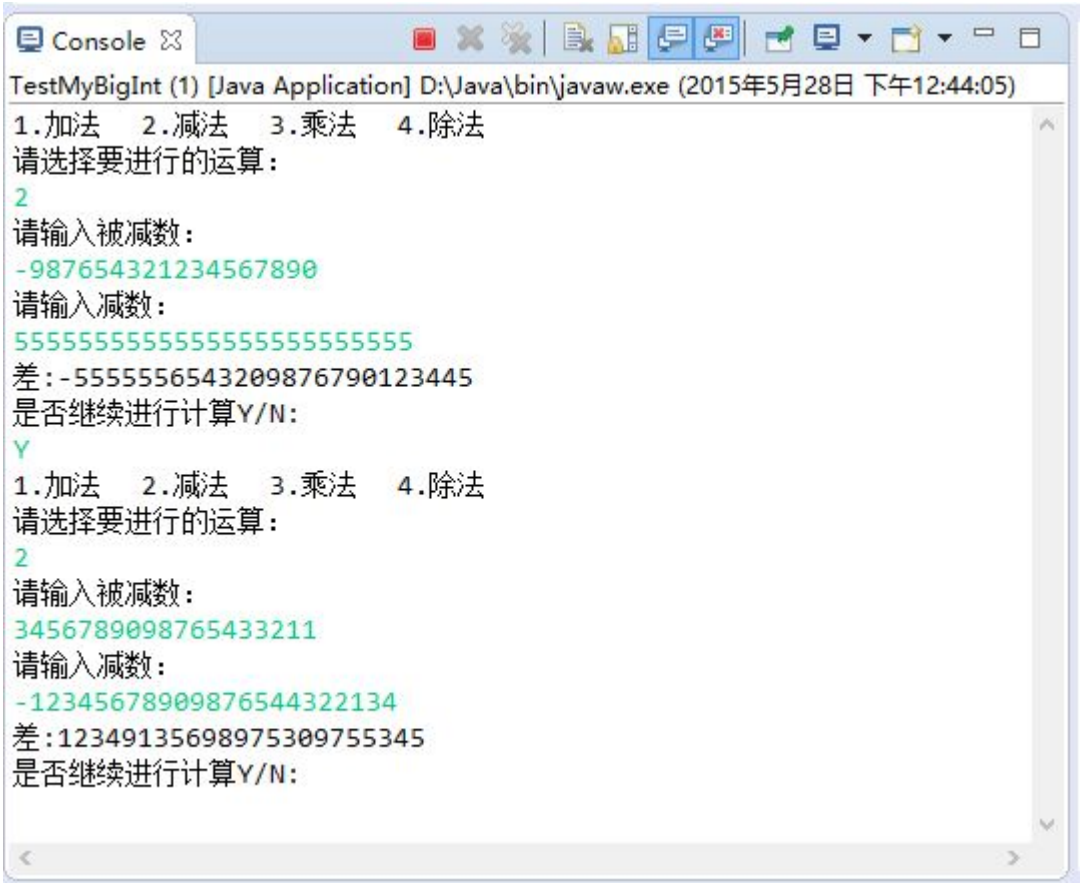
TestMyBigInt.java

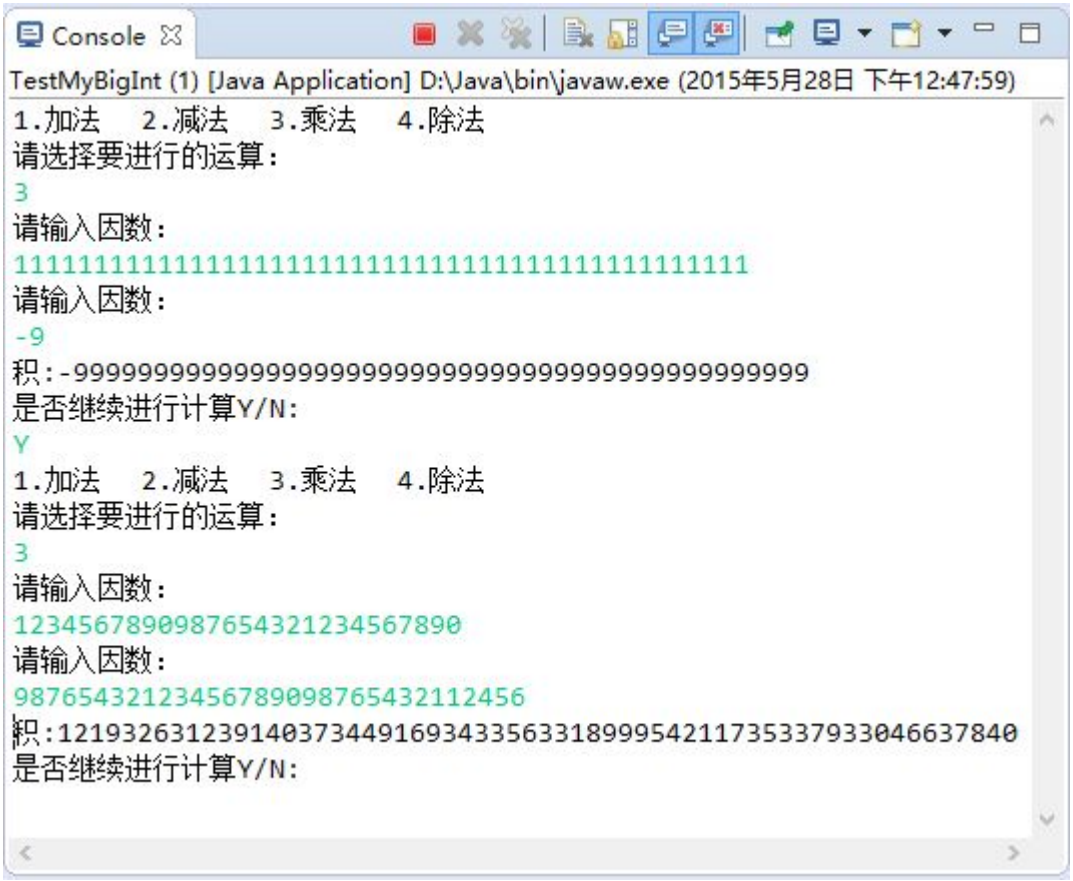
```
package Midterm;
import java.io.*;

public class TestMyBigInt {

	public static void main(String[] args) throws IOException {

		String ope=new String();
		StringBuffer para1;
		StringBuffer para2;
		int operator;


		do{
			MyBigInt BI;/**/
```

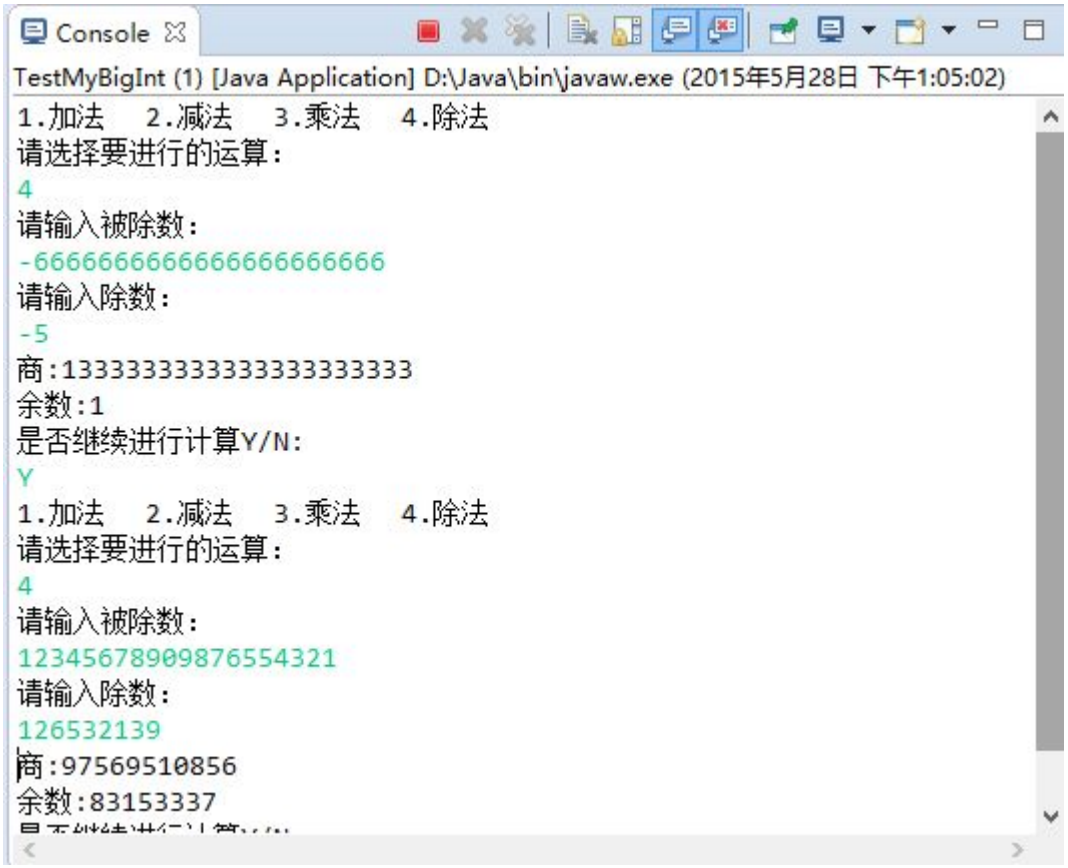
```
                System.out.println("1.加法    2.减法3.乘法4.除法");
                System.out.println("请选择要进行的运算： ");
                BufferedReader input=new BufferedReader(new
InputStreamReader(System.in));


                ope=input.readLine();
                operator=Integer.parseInt(ope);

                switch (operator){
                      case 1:
                            System.out.println("请输入加数： ");
                            ope=input.readLine();
                            para1 = new StringBuffer(ope);
                            System.out.println("请输入加数： ");
                            ope=input.readLine();
                            para2 = new StringBuffer(ope);
                            BI=new MyBigInt(para1,para2);
                            System.out.println("
和:"+BI.getAns(BI.add(BI.getBigInt1(),BI.getBigInt2())));
                            break;
                      case 2:
                            System.out.println("请输入被减数： ");
                            ope=input.readLine();
                            para1 = new StringBuffer(ope);
                            System.out.println("请输入减数： ");
                            ope=input.readLine();
                            para2 = new StringBuffer(ope);
                            BI=new MyBigInt(para1,para2);
                            System.out.println("
差:"+BI.getAns(BI.minus(BI.getBigInt1(),BI.getBigInt2())));
                            break;

                      case 3:
                            System.out.println("请输入因数： ");
                            ope=input.readLine();
                            para1 = new StringBuffer(ope);
                            System.out.println("请输入因数： ");
                            ope=input.readLine();
                            para2 = new StringBuffer(ope);
                            BI=new MyBigInt(para1,para2);
                            System.out.println("
积:"+BI.getAns(BI.multiply(BI.getBigInt1(),BI.getBigInt2())));
                            break;
                      case 4:
                            System.out.println("请输入被除数： ");
                            ope=input.readLine();
                            para1 = new StringBuffer(ope);
                            System.out.println("请输入除数： ");
                            ope=input.readLine();
                            para2 = new StringBuffer(ope);
                            BI=new MyBigInt(para1,para2);
```

```
                                BI.devide(BI.getBigInt1(),BI.getBigInt2());
                                System.out.println("
商:"+BI.getAns(BI.getQuoBigInt()));
                                System.out.println("余
数:"+BI.getAns(BI.getRemBigInt()));
                                break;

                        }
                        System.out.println("是否继续进行计算 Y/N:");
                        ope=input.readLine();
                    }while (ope.equals("Y"));
            }


}
```




Console
TestMyBigInt (1) [Java Application] D:\Java\bin\javaw.exe (2015年5月28日 下午12:44:05)
1.加法 2.减法 3.乘法 4.除法
请选择要进行的运算:
2
请输入被减数:
-987654321234567890
请输入减数:
555555555555555555555555
差:-55555565432098767901234 45
是否继续进行计算Y/N:
Y
1.加法 2.减法 3.乘法 4.除法
请选择要进行的运算:
2
请输入被减数:
3456789098765433211
请输入减数:
-12345678909876544322134
差:12349135698975309755345
是否继续进行计算Y/N:


Console
TestMyBigInt (1) [Java Application] D:\Java\bin\javaw.exe (2015年5月28日 下午12:47:59)
1.加法 2.减法 3.乘法 4.除法
请选择要进行的运算:
3
请输入因数:
1111111111111111111111111111111111111111111111
请输入因数:
-9
积:-9999999999999999999999999999999999999999999999
是否继续进行计算Y/N:
Y
1.加法 2.减法 3.乘法 4.除法
请选择要进行的运算:
3
请输入因数:
12345678909876543212345678 90
请输入因数:
98765432123456789098765432112456
积:12193263123914037344916934335633189995421173533793304663784 0
是否继续进行计算Y/N:



（2）以上述的 MyBigInt 型整数运算系统为基础，设计以 MyBigInt 型整数为系数的多项式运算系统（记为 MyPolyBigInt）

本题要求设计多项式运算系统（按一元多项式考虑），且多项式的系数为 MyBigInt 型整数，多项式的项数未知，且用户输入的多项式每一项无序，故对本题采用单链表数据结构（PolySLink 类），且其含有头节点 head。

一元多项式每一项分为指数、系数、x，故在定义单链表的节点类型时，新建一个 PolyNode 类，存储了系数域和指数域。

单链表中的节点按照指数递减排列。构造单链表时，用户每输入一项，将该新节点 v 插入（该方法 insert（v））到单链表中。插入的时候从 head 节点开始，遇到指数相同的，则将该新节点与已有的节点系数相加即可（相当于合并同类项）；若没有指数相同的，则作为新节点链接到相应位置，并保证单链表按指数从高到低排列；

有了合适的数据结构（PolyNode、PolySLink），用户输入的两个多项式，构造完后成为两个单链表——PolySLink fx,PolySLink gx;

对于 MyPolyBigInt,我们实现了多项式的加减乘功能（设结果为 ansSLink）：

（1）加法 addPoly：

先将 ansSLink 赋值为 fx，用一工作指针 PolyNode gxCurrent 分别指向 gx 的每一项，只需将 gxCurrent insert 到 ansSLink 中即可，如同在一开始接受用户的输入后构造多项式的每一项一样；

（2）减法 minusPoly：

只需将 gx 每一项系数符号*-1，调用 addPoly 即可；

（3）乘法：
将 gx 与 fx 的每一项相乘，得出的结果 insert 入 ansSLink 即可；

基于 Java interface 的 ADT 表示和 基于 Java class 的具体实现如下：

IMyPolyBigInt.java

```
package Midterm;

public interface IMyPolyBigInt {
	//返回结果
	public PolySLink getAnsSLink();

	//多项式加法
	public void addPoly(PolySLink fx,PolySLink gx);

	//多项式减法
	public void minusPoly(PolySLink fx,PolySLink gx);

	//多项式乘法
	public void multiplyPoly(PolySLink fx,PolySLink gx);
}
```

PolyNode.Java

```
//节点类
package Midterm;
public class PolyNode {
    protected BigNumStruct coe;//系数
    protected long exp;//指数
    PolyNode next;
    public PolyNode(StringBuffer s,long newExp){

         coe=new BigNumStruct(s);
         exp=newExp;
         next=null;
    }
    public PolyNode(){
      coe=null;
        exp=0;
        next=null;
    }
    public BigNumStruct getCoe() {
        return coe;
    }
    public long getExp() {
```

```
        return exp;
    }
    public PolyNode getNext() {
        return next;
    }

    public void setCoe(BigNumStruct newCoe){
      coe=newCoe;
    }

    public void setExp(long newExp){
      exp=newExp;
    }
    public void setNext(PolyNode newNext)
      {
            next=newNext;
      }

    //改变符号
    public void opposeSign(){
      coe.setSign((byte) (-1*coe.getSign()));
    }
}
```

IPolySLink.java

```
package Midterm;
//多项式链表结构
public interface IPolySLink {
      //链表是否为空
      public boolean isEmpty();

      public PolyNode getHead();

      public long getSize();

      //在 prePolyNode 后增加节点 newPolyNode
      public void addAfter(PolyNode prePolyNode,PolyNode newPolyNode);

      //将一个新节点 newPolyNode 插入到单链表中,按照指数从高到低排列
      public void insert(PolyNode newPolyNode);

      //将链表转化为多项式形式的字符串
      public String toString();
}
```

PolySLink.java

```
package Midterm;
public class PolySLink extends MyBigInt implements IPolySLink {
```

```
    protected PolyNode head;//头节点
    protected long size;//链表长度

    public PolySLink(){
          head=new PolyNode();
        size=0;
    }
    //链表是否为空
    public boolean isEmpty(){
      return size==0;
}

    public PolyNode getHead(){
          return head;
    }

    public long getSize(){
          return size;
    }


    //在 prePolyNode 后增加节点 newPolyNode
    public void addAfter(PolyNode prePolyNode,PolyNode newPolyNode){
      newPolyNode.setNext(prePolyNode.getNext());
      prePolyNode.setNext(newPolyNode);
      size++;
}

    //将一个新节点 newPolyNode 插入到单链表中,按照指数从高到低排列
    public void insert(PolyNode newPolyNode){
          PolyNode p,r;
          r=head;//r 为 p 的前驱节点
          p=head.getNext();//p 指向为当前节点
          long e=newPolyNode.getExp();
          while ((p!=null)&&(p.getExp()>=e)){
                r=p;
                p=p.getNext();
          }

          //系数相等，则指数相加，不产生新节点
          if ((e==r.getExp())&&(r!=head))
                r.setCoe(add(r.getCoe(),newPolyNode.getCoe().clone()));
          //否则插到 r 之后
          else {
                addAfter(r,newPolyNode);
          }
    }

    public String toString()
    {
          String s=new String();
          PolyNode p=head.getNext();
```

```
            while (p!=null){
                  //系数不为0需要输出
                  if (!p.getCoe().equals("0")){
                        //常数项和非常数项分别操作
                        //非常数项
                        if (p.getExp()!=0){
                              //先输符号
                              if (p.getCoe().getSign()==1) s=s+"+";
                              //非常数项的系数1不必输出
                              if (!p.getCoe().equals("1"))
s=s+p.getCoe().arrayToString();
                              s=s+"x";
                              //指数为1,1不必输出
                              if (p.getExp()!=1) s=s+"^"+p.getExp();

                        }
                        //常数项
                        else{
                              if (p.getCoe().getSign()==1) s=s+"+";

                              s=s+p.getCoe().arrayToString();
                        }

                  }
                  p=p.getNext();
            }
            if (s.equals("")) return "0";
            //首项+号不输出
            if (s.charAt(0)=='+') return s.substring(1,s.length());
            else return s;
      }

}
```

MyPolyBigInt.java

```
package Midterm;
public class MyPolyBigInt extends MyBigInt implements IMyPolyBigInt{

      protected PolySLink ansSLink;

      public MyPolyBigInt(){
            ansSLink=new PolySLink();
      }


      public PolySLink getAnsSLink(){
```

```
            return ansSLink;
      }

      //多项式加法
      public void addPoly(PolySLink fx,PolySLink gx){
            ansSLink=fx;
            PolyNode gxCurrent=gx.getHead().getNext();//gxCurrent 为 g(x)中的第一
项
            while (gxCurrent!=null){
                  ansSLink.insert(gxCurrent);
                  gxCurrent=gxCurrent.getNext();
            }
      }

      //多项式减法
      public void minusPoly(PolySLink fx,PolySLink gx){
            ansSLink=fx;
            PolyNode gxCurrent=gx.getHead().getNext();//gxCurrent 为 g(x)中的当前
项
            while (gxCurrent!=null){
                  gxCurrent.opposeSign();//改变 g(x)每一项系数的符号
                  ansSLink.insert(gxCurrent);
                  gxCurrent=gxCurrent.getNext();
            }
      }

      //多项式乘法
      public void multiplyPoly(PolySLink fx,PolySLink gx){
            PolyNode fxCurrent=fx.getHead().getNext();//fxCurrent 为 f(x)中的当前
项
            PolyNode gxCurrent=gx.getHead().getNext();//gxCurrent 为 g(x)中的当前
项

            while (fxCurrent!=null){
                  gxCurrent=gx.getHead().getNext();
                  while (gxCurrent!=null){
                        PolyNode tempAnsNode=new PolyNode();//临时存储 f(x),g(x)的
各项相乘结果

      tempAnsNode.setCoe(multiply(fxCurrent.getCoe(),gxCurrent.getCoe()));

      tempAnsNode.setExp(fxCurrent.getExp()+gxCurrent.getExp());
                        ansSLink.insert(tempAnsNode);
                        gxCurrent=gxCurrent.getNext();
                  }
                  fxCurrent=fxCurrent.getNext();
            }
      }


}
```

测试的主函数如下：

TestMyPolyBigInt.java

```
package Midterm;
import java.io.BufferedReader;
import java.io.*;
public class TestMyPolyBigInt {

	public static void main(String[] args) throws IOException {
		String ope=new String();
		long n;//n 为项数
		long exp;//exp 代表输入的指数
		long i;
		int operator;//进行的运算类型
		PolySLink fx;
		PolySLink gx;
		MyPolyBigInt MPBI;

		BufferedReader input=new BufferedReader(new
InputStreamReader(System.in));
		do{
			//构造 f(x);
			System.out.println("请输入一元多项式 f(x)的项数： ");
			fx=new PolySLink();
			ope=input.readLine();
			n=Integer.parseInt(ope);
			for (i=0;i<n;i++){
				System.out.println("系数： ");
				ope=input.readLine();
				StringBuffer coe = new StringBuffer(ope);
				System.out.println("指数： ");
				ope=input.readLine();
				exp=Integer.parseInt(ope);
				PolyNode newPN=new PolyNode(coe,exp);//构造输入的这一节点
newPN
				fx.insert(newPN);//将新输入的项插入到 fx 单链表中
			}
			System.out.println("f(x)="+fx.toString());

			//构造 g(x);
			System.out.println("请输入一元多项式 g(x)的项数： ");
			gx=new PolySLink();
			ope=input.readLine();
			n=Integer.parseInt(ope);
			for (i=0;i<n;i++){
				System.out.println("系数： ");
				ope=input.readLine();
				StringBuffer coe = new StringBuffer(ope);
				System.out.println("指数： ");
				ope=input.readLine();
				exp=Integer.parseInt(ope);
```

```
                        PolyNode newPN=new PolyNode(coe,exp);//构造输入的这一节点
newPN
                        gx.insert(newPN);//将新输入的项插入到 fx 单链表中
                  }
                  System.out.println("g(x)="+gx.toString());


                  System.out.println("1.加法    2.减法3.乘法4.除法");
                  System.out.println("请选择要进行的一元多项式运算： ");


                  ope=input.readLine();
                  operator=Integer.parseInt(ope);

                  MPBI=new MyPolyBigInt();
                  switch (operator){
                        case 1:
                              MPBI.addPoly(fx,gx);

      System.out.println("f(x)+g(x)="+MPBI.getAnsSLink().toString());
                              break;
                        case 2:
                              MPBI.minusPoly(fx,gx);
                              System.out.println("f(x)-
g(x)="+MPBI.getAnsSLink().toString());
                              break;
                        case 3:
                              MPBI.multiplyPoly(fx,gx);

      System.out.println("f(x)*g(x)="+MPBI.getAnsSLink().toString());
                              break;

                        case 4:
                              MyRatPolyBigInt MRPBI=new MyRatPolyBigInt();
                              MRPBI.devidePoly(fx, gx);
                              System.out.println("f(x)/g(x):");
                              System.out.println("商式
="+MRPBI.getQuoSLink().toString());
                              System.out.println("余式
="+MRPBI.getRemSLink().toString());
                              break;
                  }

                  System.out.println("是否继续进行计算 Y/N:");
                  ope=input.readLine();
            }while (ope.equals("Y"));
      }

}
```

测试结果：

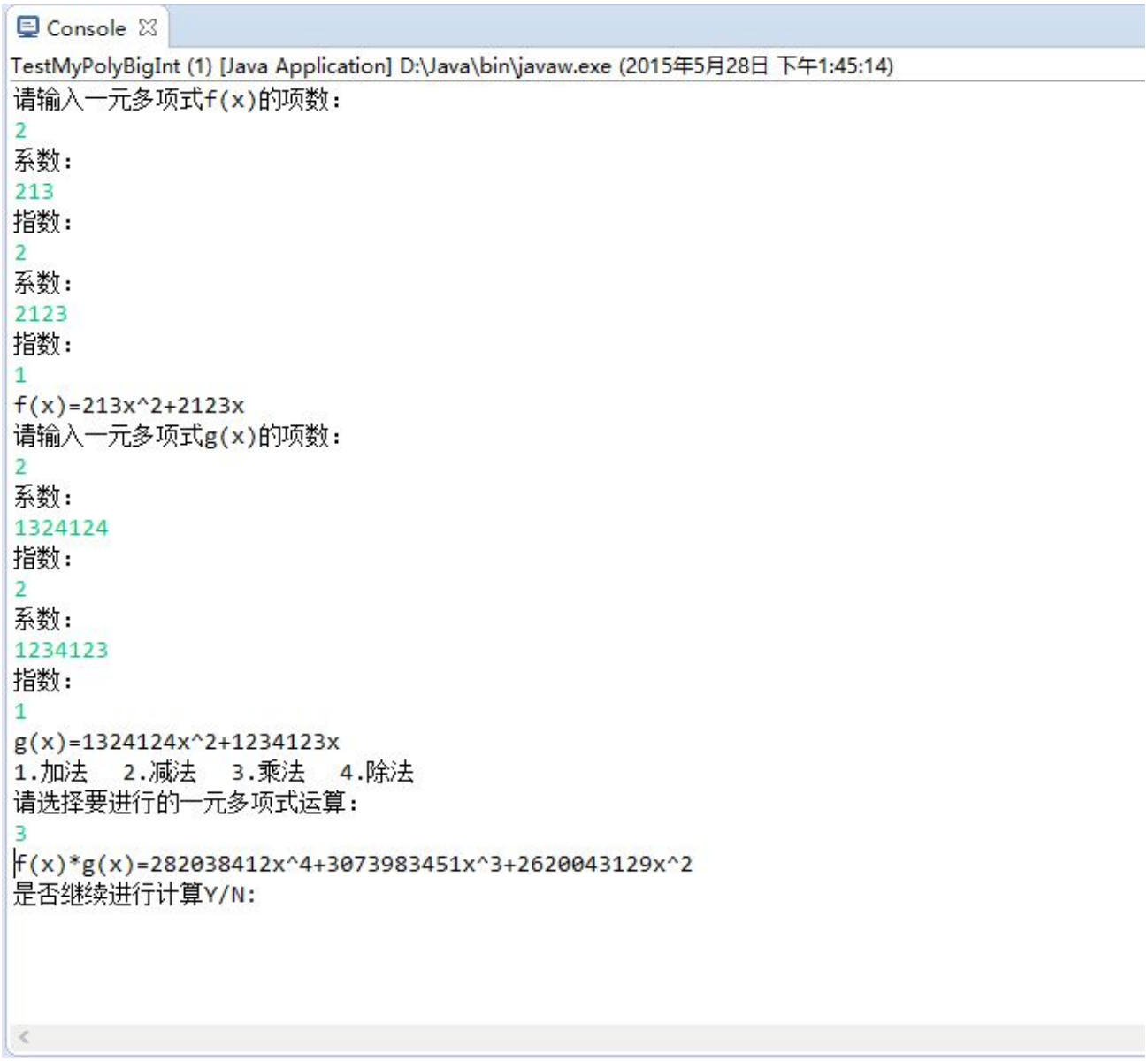
Console

```
TestMyPolyBigInt (1) [Java Application] D:\Java\bin\javaw.exe (2015年5月28日 下午1:41:23)
3
系数:
2342121342435234
指数:
3
系数:
673512471923498103274089l2
指数:
2
系数:
67
指数:
0
f(x)=2342121342435234x^3+673512471923498103274089l2x^2+67
请输入一元多项式g(x)的项数:
2
系数:
2
指数:
1
系数:
5
指数:
0
g(x)=2x+5
1.加法   2.减法   3.乘法   4.除法
请选择要进行的一元多项式运算:
1
f(x)+g(x)=2342121342435234x^3+673512471923498103274089l2x^2+2x+134
是否继续进行计算Y/N:
```

```
TestMyPolyBigInt (1) [Java Application] D:\Java\bin\javaw.exe (2015年5月28日 下午1:43:16)
请输入一元多项式f(x)的项数：
3
系数：
1
指数：
2
系数：
2
指数：
1
系数：
4
指数：
0
f(x)=x^2+2x+4
请输入一元多项式g(x)的项数：
2
系数：
71285648735124123754
指数：
132
系数：
2
指数：
0
g(x)=71285648735124123754x^132+2
1.加法    2.减法    3.乘法    4.除法
请选择要进行的一元多项式运算：
1
f(x)+g(x)=71285648735124123754x^132+2x^2+4x+8
是否继续进行计算Y/N:
```

Console

```
TestMyPolyBigInt (1) [Java Application] D:\Java\bin\javaw.exe (2015年5月28日 下午1:44:14)
请输入一元多项式f(x)的项数:
2
系数:
124123523637568
指数:
2
系数:
4375613478567813
指数:
1
f(x)=124123523637568x^2+4375613478567813x
请输入一元多项式g(x)的项数:
2
系数:
21141324
指数:
4
系数:
2
指数:
0
g(x)=21141324x^4+2
1.加法   2.减法   3.乘法   4.除法
请选择要进行的一元多项式运算:
2
f(x)-g(x)=-21141324x^4-248247047275136x^2-8751226957135626x
是否继续进行计算Y/N:
```



（3）以 MyPolyBigInt 为基础，设计有理分式运算系统(记为 MyRatPolyBigInt)。

数据结构同 MyPolyBigInt，只要需要实现分式运算系统即可，同时在做分式运算的同时会用到多项式的减法、乘法，故 MyRatPolyBigInt 需要继承 MyPolyBigInt；

由于需要返回商式和余式，在 MyRatPolyBigInt 增加两个全局变量 PolySLink quoSLink,remSLink，分别存储商式和余式；

分式运算系统具体实现如同笔算多项式的除法：

1.若被除式的最高项 fxCurrent 次数大于等于除式的最高项 gxCurrent 系数，则新建一个 PolyNode tempQuoNode，其系数为 fxCurrent 和 gxCurrent 的系数之商，其次数为 fxCurrent 和 gxCurrent 的次数之差；

2.新建一条 PolySLink tempSLink，只含 head 和 tempQuoNode 两个节点的临时链，调用 multiplyPoly 方法，使得 tempQuoNode 遍乘除式 gx 的每一项，并得到一个临时积；

3. 被除式与临时积之差作为新的被除式 fx，重复 1；
4. 结束 while 循环后，fx 即为余式，quoSLink 即为商式；

（程序中对于系数相除采用了 MyBigInt 的 **public void** devide(BigNumStruct bigInt1,BigNumStruct bigInt2)，并未使用带小数的除法）：
基于 Java interface 的 ADT 表示和 基于 Java class 的具体实现如下：

IMyRatPolyBigInt.java

```
package Midterm;
public interface IMyRatPolyBigInt {
	//得到商式
	public PolySLink getQuoSLink();

	//得到余式
	public PolySLink getRemSLink();

	//多项式除法运算
	public void devidePoly(PolySLink fx,PolySLink gx);
}
```

MyRatPolyBigInt.java

```
package Midterm;
public class MyRatPolyBigInt extends MyPolyBigInt{
	protected PolySLink quoSLink,remSLink;

	public MyRatPolyBigInt(){
		quoSLink=new PolySLink();
		remSLink=new PolySLink();
	}
	//得到商式
	public PolySLink getQuoSLink(){
		return quoSLink;
	}
	//得到余式
	public PolySLink getRemSLink(){
		return remSLink;
	}

	//多项式除法运算
	public void devidePoly(PolySLink fx,PolySLink gx){

		PolyNode fxCurrent=fx.getHead().getNext();//fxCurrent 为 f(x)中的当前
项
		PolyNode gxCurrent=gx.getHead().getNext();//gxCurrent 为 g(x)中的当前
项
```

```
            //PolyNode quoCurrent=quoSLink.getHead();//quoCurrent 为商的当前项

            PolyNode tempQuoNode;//临时存储商的一项
            MyBigInt BI=new MyBigInt();
            while (fxCurrent.getExp()>=gxCurrent.getExp()){
                  tempQuoNode=new PolyNode();
                  BI.devide(fxCurrent.getCoe(),gxCurrent.getCoe());
                  tempQuoNode.setCoe(BI.getQuoBigInt());//被除式的最高项系数除以除
式的最高项系数为商的最高项系数
                  tempQuoNode.setExp(fxCurrent.getExp()-gxCurrent.getExp());//被
除式的最高项指数减去除式的最高项指数为商的最高项指数
                  quoSLink.insert(tempQuoNode);//将该临时点放入商中

                  //构造一条只含 head 和 tempQuoNode 的单链表
                  PolySLink tempSLink = new PolySLink();
                  tempSLink.insert(tempQuoNode);

                  //临时商遍乘除数
                  super.ansSLink=new PolySLink();
                  super.multiplyPoly(gx, tempSLink);

                  //使用 tempProduct 存储遍乘结果
                  PolySLink tempProduct=new PolySLink();
                  tempProduct=super.getAnsSLink();

                  //当前被除式-遍乘结果=新的被除式
                  super.ansSLink=new PolySLink();
                  super.minusPoly(fx,tempProduct);
                  fx=super.getAnsSLink();//新的被除式
                  fxCurrent=fx.getHead().getNext();//新的被除式的第一项，但此项的系
数一定为 0；

                  while ((fxCurrent!=null) &&
fxCurrent.getCoe().arrayToString().equals("0") )
fxCurrent=fxCurrent.getNext();//系数为 0，则后移
                  if (fxCurrent==null) break;
            }
            remSLink=fx;

      }
}
```

测试的主函数同（2）题中的 **TestMyPolyBigInt.java;**

测试结果：

五、实验体会

1.大整数的运算基本思路即为将大整数分段，这样每一段都为基本类型，对每一段没名分别进行运算，再将计算出的每一段结果合并即为最终结果；

2.存储大数有链式和顺序存储方式两种方法。链式可以适应不定长度的大整数，但由于存储空间是离散的，故随机访问效率不高；顺序存储主要是数组方式，其存储空间是连续的，随机访问率高，一般数组在定义的时候即为其分配指定长度的内存空间，同时为了使数组适应不定长度的大整数，我们可以根据输入的参与运算的参数的长度，再开辟相应长度的数组来存储大数。

3.将大整数分段后，对每一段储存，一般使用数组储存，可以每一个数组单元储存一个十进制位，但这样对空间利用率和时间效率上都不高。本实验报告中采用了十亿进制，每九位储存在一个数组单元中，提高了空间利用率和时间效率；

另外也有采用计算机的字长进制数组来存储大数，如 2^{16} 进制，2^{32} 进制，这样通过移位操作能达到基本运算的目的，时间效率会高很多，但是涉及到进制转换会稍复杂；

4.在对大数结果进行输出时，对每一个数组单元的内容进行输出，但往往会出项里面的内容不足位，例如本实验报告采用的是十亿进制，一数组单元存储着 4587，则输出时并不直接输出 4587，而应输出 000004587；

5.用户输入的大整数我们使用字符串储存，后期对字符串需要改动（如清 0、替换、改动符号等），而 String 为 final 类，对其改动是不可行的，故在存储大整数的字符串形式时，我们使用了可变字符串类型 SringBuffer 类，对 StringBuffer 类操作不像 String 直接，往往要 StringBuffer 的方法；

4.对两个对象间用“=”赋值，一般不是内容的复制，而是地址的赋值，使两个对象指向的是同一个内存地址，一者对该内存内容进行了改动，另一对象中访问该内容，会发现同时发生了变动。所以有时为了实现内容的复制，需要使用 clone（）方法；

6.对于结果是带小数的除法，结果分为整数位和小数点位，在调用完 MyBigInt 类的 devide 方法后，商和余数均已求出，整数位即为商；

对于小数位，只需将余数的末尾补 0 至与除数一样长度，保留 k 位小数，则继续在末尾添 k 个 0，将添完 0 后的余数作为新的被除数，除数不变，再调用 MyBigInt 类的 devide 方法，结果的商即为小数位；

7.多项式的存储使用了单链表，是考虑到用户输入时的无序性；而通过使用 insert（v）方法，可实现合并同类项的功能或新增节点的功能；

8.多项式的计算上，分为指数的操作和系数的操作，指数由于为基本数据类型，可直接进行基本运算，而系数为大数类型，应当按照 MyBigInt 类中的对应方法来进行运算；

六、参考文献

1. 主讲课英文教材 **Goodrich, Tamassia: *Data Structures and Algorithms in Java, 5th Edition International Student Version chapter 3***

2. 实验教材：汪萍，陆正福等编著 数据结构与算法的问题与实验 第 1 章

3.（如有其它参考文献，请列出）