《数据结构与算法实验》第 3 次实验

学院： 专业： 年级：
姓名： 学号： 日期： 2022 年 3 月 14 日

第一部分 基础实验

一、 实验目的

1. 学习掌握线性表，尤其是单链表的相关知识，熟练运用顺序表的实现和逆置等操作。

2. 通过线性表的应用实现实验的交与并。

二、 实验内容

1. **单链表及其相关操作（验证实验）**：实验书 p174：单链表的实现。

- 加入求线性表的长度等操作：结合本章习题的算法设计题做。
- 重新给定测试数据，验证抛出异常机制。
- 将单链表的结点结构用结点类实现。

2. **单链表的设计实现（不带头节点）**： 参考实验书 p174 单链表的实现做法，实现不带头节点的单链表。

- 加入求线性表的长度等操作：结合本章习题的算法设计题做。
- 重新给定测试数据，验证抛出异常机制。
- 将单链表的结点结构用结点类实现。

3. **顺序表的应用，实现集合的“并”和“交”（设计实验）**：

- 有两个顺序表 LA 和 LB，把它们当成集合来使用，考虑它们的并运算和交运算。可以把顺序表当作一个抽象数据类型，直接利用它的定义来实现要求的运算。

三、 设计与编码

1. 本实验用到的理论知识

- **顺序表逆置算法**：利用数组逆置法实现数组向左边移动 p 位。分别令 i 指向顺序表的表头，j 指向表尾，交换 i，j 位置上的两个数，然后利用 i++, j--, 直到 $i \geq j$。

- **单链表逆置算法**：改变指针 next 的指向。

```
void Reverse()
{
    p = first, before = NULL;
    while (p != NULL)
    {
        tnext = p->next;
        p -> next = before;
        before = p;
        p = tnext;
    }
    first = before;
}
```

- **单链表去重算法：**从头开始往后找，把后面相同的值都删掉。

```
void Purge()
{
    p = first;
    while(p!=NULL)
    {
        q=p->next,before=p;
        while(q!=NULL)
        {
            if(q->data==p->data)
            {
                s=q->next;
                before->next=s;
                delete q;
                q=s;
                continue;
            }
            before=q;
            q=q->next;
        }
        p=p->next;
    }
}
```

·集合的交算法：对于已知的链表 LA、LB。逐个考察 LA 中的元素，如果它不在 LB 中，则将它从 LA 中删去。对每个元素遍历操作后得到的 LA 为交集。

·集合的并算法：对于已知的链表 LA， LB。逐个考察 LB 中的元素，如果它不在 LA 中，则将它插入 LA。对每个元素遍历操作后得到的 LA 为并集。

2. 算法设计

（1） **不带头节点的单链表：**定义单链表类，设置头指针。

设计成员函数：构造函数（有参、无参）、析构函数、长度函数、位置函数、插入函数、删除函数、输出函数、复制构造函数、重载运算符、判断是否递增函数、逆置函数、按大小插入函数、去重函数（具体函数参见编码）.

（2） **集合的交与并：**定义顺序表类。

设计成员函数：构造函数（有参、无参）、析构函数、返回长度的函数、输出函数、并集函数（友元）、交集函数（友元）.

3. 编码

(1) 单链表及其相关操作（验证实验）-LinkList.h/ LinkList.cpp/ LinkList_main.cpp

编码均与课本一致，故在此不作赘述。

(2) 单链表的设计实现（不带头节点）- LinkList.h/ LinkList.cpp/ LinkList_main.cpp

在(1)的基础上修改模板类中 LinkList 无参构造函数，同时在所有成员函数中针对头结点进行相应修改即可，注意使用 new 和 delete。

```
#ifndef LinkList_H
#define LinkList_H

template <class DataType>
class Node
{
    public:
        DataType data;
        Node<DataType> *next;
};

template <class DataType>
class LinkList
```

```
{
    public:
        LinkList( );
        LinkList(DataType a[ ], int n);
        ~LinkList( );
        int Locate(DataType x);
        void Insert(int i, DataType x);
        DataType Delete(int i);
        void PrintList( );
        LinkList(LinkList<DataType> &L);
        LinkList<DataType> &  operator=(LinkList<DataType> &L
);
        Node<DataType> *getfirst();
        int Length( );
        int Increase( );
        void Reverse( );
        void Insert(DataType x);
        void Purge();

    private:
        Node<DataType> *first;
};
#endif
```

(2) 单链表的设计实现（不带头节点）- LinkList.cpp

```
#include <iostream>
#include "LinkList.h"
using namespace std;

template <class DataType>
LinkList<DataType> :: LinkList( )
{
    first = NULL;
}

template <class DataType>
LinkList<DataType> :: LinkList(DataType a[ ], int n)
{
    Node<DataType> *r, *s;
    first = new Node<DataType>;
    first->data =a[0];
    r = first;
```

```
    for (int i = 1; i < n; i++)
    {
        s = new Node<DataType>;
        s->data = a[i];
        r->next = s;
        r = s;
    }
    r->next = NULL;
}

template <class DataType>
LinkList<DataType> :: ~LinkList( )
{
    Node<DataType> *q;
    while (first != NULL)
    {
        q = first;
        first = first->next;
        delete q;
    }
}

template <class DataType>
LinkList<DataType>::LinkList(LinkList<DataType> &L)
{
    Node<DataType> *s,*srcptr=L.first;
    Node<DataType>  *destptr=first=new Node<DataType>;
    while (srcptr->next!=NULL)
    {
          s=new Node<DataType>;
          s->data=srcptr->next->data;
          destptr->next=s;
          destptr=destptr->next;
          srcptr=srcptr->next;
     }
     destptr->next=NULL;
    Node<DataType> *tt=first;
    first=first->next;
    delete tt;
}
```

```
template <class DataType>
LinkList<DataType>& LinkList<DataType>::operator=(LinkList<
DataType> &L)
{
    Node<DataType> *s,*srcptr=L.first;
    Node<DataType>  *destptr=first=new Node<DataType>;
    while (srcptr->next!=NULL)
    {
        s=new Node<DataType>;
        s->data=srcptr->next->data;
        destptr->next=s;
        destptr=destptr->next;
        srcptr=srcptr->next;
    }
    destptr->next=NULL;
    Node<DataType> *tt=first;
    first=first->next;
    delete tt;
    return *this;
}

template <class DataType>
void LinkList<DataType> :: Insert(int i, DataType x)
{
    if(i==1)
    {
        Node<DataType> *p,*s;
        s = new Node<DataType>;
        s->data = x;
        s->next = first->next;
        first = s;
    }
    else
    {
        Node<DataType> *p = first, *s;
        int count = 1;
        while (p != NULL && count < i - 1)
        {
            p = p->next;
            count++;
        }
        if (p == NULL) throw i;
```

```
        else
        {
            s = new Node<DataType>;
            s->data = x;
            s->next = p->next;
            p->next = s;
        }
    }
}
template <class DataType>
void LinkList<DataType> :: Insert(DataType x)
{
    Node<DataType> *p = first,*s;
    while(p!=NULL)
    {
        if(p->next==NULL)
        {
            s=new Node<DataType> ;
            s->data=x;
            s->next=NULL;
            p->next=s;
            return ;
        }
        if((p->data<=x)&&(p->next->data>x))
        {
            s=new Node<DataType> ;
            s->data=x;
            s->next=p->next;
            p->next=s;
            return ;
        }
        p=p->next;
    }
}

template <class DataType>
DataType LinkList<DataType> :: Delete(int i)
{
    Node<DataType> *p, *q;
    DataType x;
    if(i==1)
```

```
    {
        p=first;
        first=first->next;
        x=p->data;
        delete p;
        return x;
    }
    else
    {
        int count = 1;
        p = first;
        while (p != NULL && count < i - 1)
        {
            p = p->next;
            count++;
        }
        if (p == NULL || p->next == NULL)
            throw i;
        else
        {
            q = p->next; x = q->data;
            p->next = q->next;
            delete q;
            return x;
        }
    }
}

template <class DataType>
void LinkList<DataType> :: Purge()
{
    Node<DataType> *p = first,*q,*s,*before;
    while(p!=NULL)
    {
        q=p->next;
        before=p;
        while(q!=NULL)
        {
            if(q->data==p->data)
            {
                s=q->next;
```

```
                before->next=s;
                delete q;
                q=s;
                continue;
            }
            before=q;
            q=q->next;
        }
        p=p->next;
    }
}
template <class DataType>
int LinkList<DataType> :: Locate(DataType x)
{
    Node<DataType> *p = first;
    int count = 1;
    while (p != NULL)
    {
        if (p->data == x) return count;
        p = p->next;
        count++;
    }
    return 0;
}

template <class DataType>
void LinkList<DataType> :: PrintList( )
{
    Node<DataType> *p = first;
    while (p != NULL)
    {
        cout << p->data<<"  ";
        p = p->next;
    }
    cout<<endl;
}

//Addtional functions:
template <class DataType>
int LinkList<DataType> ::Length( )
{
    Node<DataType> *p = first;
```

```
    int l=0;
    while(p!=NULL)
    {
        l++;
        p=p->next;
    }
    return l;
}

template <class DataType>
Node<DataType> *  LinkList<DataType> ::getfirst()
{    return   first;
}

template <class DataType>
int LinkList<DataType> :: Increase( )
{
    Node<DataType> *q, *p = first;
    while (p->next != NULL)
    {   q=p->next;
        if(p->data<q->data)  p=q;
        else return 0;
    }
    return 1;
}

template <class DataType>
void LinkList<DataType> ::Reverse()
{
    Node<DataType> *p = first;
    Node<DataType> *before=NULL,*tnext;

    while (p != NULL)
    {
        tnext=p->next;
        p->next=before;
        before=p;
        p =tnext;
    }
    first=before;
}
```

(2)单链表的设计实现（不带头节点）- LinkList_main.cpp

```
#include<iostream>
#include"LinkList.cpp"
using namespace std;

int main( )
{
    int r[5]={1, 9, 3, 4, 5};
    LinkList<int> L(r, 5),L2(r, 5);

    if(L.Increase()==1)
        cout<<"The LinkList is incresing."<<endl;
    else
        cout<<"The LinkList is not incresing."<<endl;

    cout<<"Before inserting£º"<<endl;
    L.PrintList( );
    try
    {
        L.Insert(2, 3);
    //  L.Insert(22, 3);
        cout<<"After inserting 3£º"<<endl;
        L.PrintList( );
    }
    catch (int k )
    {
        cout<<"Insert error! i="<<k<<endl;
    }

    cout<<"After inserting 5£º"<<endl;
    L.PrintList( );
    cout<<"The position of 5 is:"<<endl;
    cout<<L.Locate(5)<<endl;
    cout<<"Before deleting the first element of the LinkList£º"<<
endl;
    L.PrintList( );
    try
    {
        L.Delete(1);
    //  L.Delete(66);
        cout<<"After deleting£º"<<endl;
```

```
        L.PrintList( );
    }
    catch (int k)
    {
        cout<<"Delete error! i="<<k<<endl;
    }

    cout<<"The length is:"<<endl<<L.Length()<<endl;
    L2.Reverse();
    cout<<"The reversed LinkList is:"<<endl;
    L2.PrintList();

    cout<<"Before deleting the same elements£º"<<endl;
    int r1[9]={1, 2, 4,4,4,5, 5, 6,6};
    LinkList<int> L1(r1, 9);
    L1.PrintList();
    L1.Purge();
    cout<<"After deleting the same elements£º"<<endl;
    L1.PrintList();
    L1.Insert(3);
    cout<<"Insert 3 to build an orderly LinkList£º"<<endl;
    L1.PrintList();
    return 0;
}
```

(3)集合的交与并-Union-Intersection.cpp

```
#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;
const int MaxSize=100;
template <class DataType>
class SeqList
{
    public:
        SeqList () ;
        SeqList (DataType a[], int n);
        ~SeqList(){ ;}
        int Length() {return length;}
```

```
        void Insert (int i, DataType x);//插入
        int Locate (DataType x);
        void PrintList();                //输出
        void Delete (int i);             //删除
        template <class T>
        friend void Union (SeqList<T>& LA, SeqList<T>& LB );
        template <class T>
        friend void Intersection( SeqList<T> & LA, SeqList<T> & LB
 );
    private:
        DataType data[MaxSize];
        int length;
};

template <class DataType>
SeqList<DataType> ::SeqList()
{
    length=0;
}

template <class DataType>
SeqList<DataType> ::SeqList(DataType a[],int n)
{
    if (n> MaxSize) throw "参数非法";
    for(int i=0;i<n;i++)
        data[i]=a[i];
    length= n;
}

template < class DataType>
void SeqList< DataType> ::PrintList()
{
    for(int i=0; i< length;i++ )
        cout<<data[i]<<" ";
    cout<<endl;
}

template <class DataType>
void SeqList<DataType> ::Insert (int i,DataType x)
{
    if (length>=MaxSize)
        throw "上溢";
    if(i< 1||i> length+1)
        throw "位置异常";
    for (int j=length;j>=i;j--)
```

```
        data[j]= data[j-1];
    data[i-1]=x;
    length++;
}

template < class DataType>
void SeqList<DataType> ::Delete (int i)
{
    if(length==0) throw "下溢";
    if(i<1||i>length ) throw  "位置";
template < class T>
void Intersection ( SeqList<T> & LA,     SeqList<T> & LB )
{
    int n1 = LA.Length ( );
    int k, i = 0;
    T x;
    while ( i < n1 )
    {
        x = LA.data[i];  //在LA中取一元素
        k = LB.Locate(x);//在LB中搜索它
        if (k == 0)      //若在LB中未找到
        {
            LA.Delete(i+1);
            n1--;
        }               //在LA中删除它
       else i++;
    }
 }

int main()
{
    int ta[1000],tn;
    cout<<"请输入顺序表LA长度(不超过100): "<<endl;
    cin>>tn;
    cout<<"请逐个输入顺序表LA中的数字（整数）"<<endl;
    for(int i=0;i<tn;i++)
        cin>>ta[i];
    SeqList<int> A2(ta,tn);
    SeqList<int> A1(ta,tn);

    cout<<"请输入顺序表LB长度(不超过100): "<<endl;
    cin>>tn;
    cout<<"请逐个输入顺序表LB中的数字（整数）"<<endl;
    for(int i=0;i<tn;i++)
        cin>>ta[i];
```

```
    SeqList<int> B2(ta,tn);
    SeqList<int> B1(ta,tn);

    Union(A1,B1);
    Intersection(A2,B2);
    cout<<"LA与LB的并为: "<<endl;
    A1.PrintList();
    cout<<"LA与LB的交为: "<<endl;
    A2.PrintList();
    return 0;
}
```

注：在此未建立项目，而是把所有需要使用的自定义函数统一写在同一个 cpp 文件内，故文件较长。

四、 运行与测试

1. 单链表及其相关操作（验证实验）

```
The LinkList is not incresing.
Before inserting:
1  9  3  4  5
After inserting 5:
1  3  9  3  4  5
The position of 5 is:
6
Before deleting,the LinkList is:
1  3  9  3  4  5
After deleting:
3  9  3  4  5
The length is:
5
The reversed LinkList is:
5  4  3  9  1
Before deleting the same elements:
1  2  4  4  4  5  5  6  6
After deleting the same elements:
1  2  4  5  6
Insert 3 to build an orderly LinkList:
1  2  3  4  5  6
请按任意键继续. . .
```

2.单链表的设计实现（不带头节点）

由于实验 1.2 只在程序的设计上有区别，而在结果的实现上完全一致，所以输出的结果和 1.的也完全相同。

3.顺序表的应用，实现集合的“并”和“交”（设计实验）

```
请输入顺序表LA长度(不超过100)：
10
请逐个输入顺序表LA中的数字（整数）
283 912 234 532 239 91 17 296 26 88
请输入顺序表LB长度(不超过100)：
8
请逐个输入顺序表LB中的数字（整数）
123 932 12 91 88 349 454 17
LA与LB的并为：
283 912 234 532 239 91 17 296 26 123 932 12 349 454 88
LA与LB的交为：
91 17 88
请按任意键继续. . .
```

五、 总结与心得

通过这次实验，我对顺序表和模板有了更深刻的理解和更成熟的运用，对类中的函数操作也更加得心应手。虽然对于有头结点和无头结点分别进行程序设计还是有些心里上的恐惧，对插入、删除操作的验证还有些薄弱，在这学期接下来的时间里希望可以有所突破。

第二部分 综合实验：大整数运算（实验书 P182）

一、问题描述

C++ 语言中的 int 类型能表示的整数范围是-2^{31}到$2^{31}-1$。即使是对于 long long 类型，其范围也只有-2^{63}到$2^{63}-1$。而有些问题需要处理的整数远远不止 20 位，这种大整数，用 C++语言的基本数据类型无法直接表示。因此，我们需要设计另外的算法，进行两个大整数 A,B 的加、减、乘、除等基本的四则运算。

二、基本要求

- 大整数的长度在 100 位以下，且默认输入的大整数均为正数；
- 设计存储结构表示大整数；

- 设计算法实现两个大整数的加、减、乘和除四种基本的代数运算；
- 分析算法的时间复杂度和空间复杂度。

三、设计思想

处理大整数的一般方法是用数组存储大整数,即开一个比较大的整型数组，数组元素代表大整数的一位,通过数组元素的运算模拟大整数的运算。根据计算的方便性决定将大整数由低位到高位还是由高位到低位存储到数组中，例如乘法是由低位到高位进行运算,并且可能要向高位产生进位,所以应该由低位到高位存储。如果从键盘输入大整数，一般用字符数组存储,这样无需对大整数进行分段输入,当然输入到字符数组后需要将字符转换为数字。

储存大整数 A，实际上是要储存 A 的每一位数字。每一位数字的取值范围是 0 到 9 的整数，因此我们采用顺序表对数组进行储存和提取。

四、算法设计

1.大整数的输入

先初始化一个数组 a[100]用于存放大整数，此时默认输入的大整数不超过 100 位，如果希望整数更大则可以在初始化数组时进行改变。用字符串 s 写入大整数 A，与此同时计算大整数的长度 l，再逆序从低位到高位存入顺序表的数组 a[1]中。

2.判断函数（BigInt_Judge）

即判断输入的 A、B 两个大整数的大小。

由于 A、B 已经被存入数组，则可以先从它们的长度进行比较，如果相同再从最高位开始逐位比较，直到出现大小关系为止。

比较结果：如果 A＞B，返回 1；A＜B，返回-1；A、B 每一位都相同，即 A=B，返回 0.

3.退位函数（BigInt_Abdicate）

用于判断整数的位数的函数。

对于大整数 A，如果它的第 m 位是 0，则说明这意味不是最高位，转入判断 m=m-1 位是否为 0；如果第 m 位非零，则 m 就是整数 A 的长度；如果 A 中每一位数字都为 0，则说明这个大整数的值就是 0，此时修改大整数 A 的长度为 1.

4.加法函数（BigInt_ADD）

将 A、B 两个大整数逐位相加，存入数组 C，即 $c_i=a_i+b_i$.对于 C 中元素，在进行加法运算

中可能大于 10，故让每一位都对 10 做除法，整数商部分进位到 c_{i+1}，余数重新储存在 c_i. 最后使用退位函数对结果 C 的位数进行判断。

5. 减法函数（BigInt_SUB）

先使用判断函数判断大整数 A、B 的大小，如果 A＞B 则直接相减；如果 A＜B 则将较大数储存为 A，较小数储存为 B，最后在结果加上负号。对 A、B 两个大整数进行逐位相减，储存进结果数组 C，最后使用退位函数对结果 C 的位数进行判断。

6. 乘法函数（BigInt_MUL）

模仿乘法的竖式对大整数乘法进行程序设计。

对于大整数 A、B，将 A 的每一位和 B 的每一位相乘，得到新的数组 c[i+j]，对于数组 C 中的每一位，c_i 可能大于 10，故让每一位都对 10 做除法，整数商部分进位到 c_{i+1}，余数重新储存在 c_i. 最后使用退位函数对结果 C 的位数进行判断。

7. 除法函数（BigInt_DIV）

先使用判断函数判断大整数 A、B 的大小，如果 A＜B，则商为 0，余数为 A，结束算法。

当 A≥B 时，模仿除法的竖式对大整数除法进行程序设计。先从被除数 A 的高位除起，除数 B 有几位，就看 A 的前几位。如果不够除，就多看一位。然后对于这前几位的构成的数，用它除以 B。（实际上用的是这个数不停的减 B 直至不够减）。除到 A 的哪一位，就把商写在哪一位的上面；如果不够除，就在这一位上商 0。除得的余数必须比除数小。

五、代码实现

1. 设计头文件

```
const int MaxSize = 1000;
template <class DataType>
class SeqList
{
    public:
        SeqList( ) ;
        SeqList(DataType a[ ], int n);
        ~SeqList( ) ;
        int Length( ) ;
        void PrintList( );
        void negtive(); //用于设置结果的正负
        //判断函数：用于判断A，B大小
        template <class T>
        friend  int BigInt_Judge(SeqList<T> A, SeqList<T> B);
        //退位函数：推掉减法或乘法高位多于的0
        template <class T>
        friend void BigInt_Abdicate(int l,SeqList<T> &C);
```

```
        //加法函数
        template <class T>
        friend void BigInt_ADD(SeqList<T> A, SeqList<T> B,SeqList<T> &C);
        //减法函数
        template <class T>
        friend void BigInt_SUB(SeqList<T> A, SeqList<T> B,SeqList<T> &C);
        //绝对值C=A-B函数，默认A>=B
        template <class T>
        friend void BigInt_ABS(SeqList<T> A, SeqList<T> B,SeqList<T> &C);
        //乘法函数
        template <class T>
        friend void BigInt_MUL(SeqList<T> A, SeqList<T> B,SeqList<T> &C);
        //除法函数
        template <class T>
        friend void BigInt_DIV(SeqList<T> A, SeqList<T> B,SeqList<T> &C,SeqList<T> &D);
    private:
        DataType data[MaxSize];             //存放数据元素的数组
        int length;                         //线性表的长度
        int sign;                           //符号
};
```

2. 设计成员函数

```
#include "SeqList.h"
#include <iostream>
#include <cstdio>
using namespace std;

template <class DataType>
int SeqList<DataType> ::Length( )
{
    return length;
}

template <class DataType>
SeqList<DataType> :: SeqList()
{
    length = 0;
    for (int i = 0; i < MaxSize; i++)
        data[i] = 0;
    sign=1;
}

template <class DataType>
SeqList<DataType> :: SeqList(DataType a[ ], int n)
{
    if (n > MaxSize) throw "参数非法";
    for (int i = 0; i < n; i++)
        data[i] = a[i];
```

```
    length = n;
    for (int i = n; i < MaxSize; i++)
        data[i] = 0;
    sign=1;
}

template <class DataType>
SeqList<DataType> :: ~SeqList(){ }

template <class DataType>
void SeqList<DataType> :: negtive()
{
    sign=-1;
}

template <class DataType>
void SeqList<DataType> :: PrintList( )
{
    if(sign==-1) cout<<"-";
    for (int i = length-1; i >=0; i--)
        cout<<data[i];
    cout<<endl;
}

template <class T>
void BigInt_Abdicate(int l,SeqList<T> &C)
{
    while(C.data[l-1]==0&&l>0)
        l--;
    if(l==0) l++;
    C.length=l;
}

template <class T>
int BigInt_Judge(SeqList<T> A, SeqList<T> B)
{
    int l1=A.length,l2=B.length;
    if(l1<l2) return -1;
    else if(l1>l2) return 1;
    else
    {
        for(int i=l1-1;i>=0;i--)
        {
            if(A.data[i]<B.data[i])
                return -1;
            if(A.data[i]>B.data[i])
                return 1;
        }
    }
    return 0;
}

template <class T>
void BigInt_ADD(SeqList<T> A, SeqList<T> B,SeqList<T> &C)
```

```
{
    int n1=A.length,n2=B.length,n=max(n1,n2);
    for(int i=0;i<n;i++)
        C.data[i]=A.data[i]+B.data[i];
    for(int i=0;i<n;i++)
    {
        C.data[i+1]=C.data[i+1]+C.data[i]/10;
        C.data[i]=C.data[i]%10;
    }
    BigInt_Abdicate(n+1,C);
}

template <class T>
void BigInt_ABS(SeqList<T> A, SeqList<T> B,SeqList<T> &C)
{
    int l1=A.length;
    for(int i=0;i<l1;i++)
    {
        if(A.data[i]<B.data[i])
        {
            A.data[i+1]--;
            A.data[i]=A.data[i]-B.data[i]+10;
        }
        else
        {
            A.data[i]=A.data[i]-B.data[i];
        }
        C.data[i]=A.data[i];
    }
    BigInt_Abdicate(l1,C);
}

template <class T>
void BigInt_SUB(SeqList<T> A, SeqList<T> B,SeqList<T> &C)
{

    int t=BigInt_Judge(A,B);
    if(t==-1)
    {
        C.negtive();
        BigInt_ABS(B,A,C);
    }
    else
    {
        BigInt_ABS(A,B,C);
    }
}

template <class T>
void BigInt_MUL(SeqList<T> A, SeqList<T> B,SeqList<T> &C)
{
    int l1=A.length,l2=B.length;

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

```
            for(int j=0;j<l1;j++)
                C.data[i+j]=C.data[i+j]+B.data[i]*A.data[j];

        for(int i=0;i<l1+l2;i++)
        {
            C.data[i+1]=C.data[i+1]+C.data[i]/10;
            C.data[i]=C.data[i]%10;
        }
        BigInt_Abdicate(l1+l2,C);
    }

    template <class T>
    void BigInt_DIV(SeqList<T> A, SeqList<T> B,SeqList<T> &C,SeqList<T> &D)
    {
        int t=BigInt_Judge(A,B);
        if(t==-1) //A<B
        {
            D=A;//余数为A
            C.length=1;
            C.data[0]=0;//商为0
        }
        else//A>=B
        {
            int l1=A.length;
            int i=l1-1;
            SeqList<int> tA;
            for(;i>=0;i--)
            {
                tA.length++;
                for(int j=tA.length-1;j>=1;j--)
                {
                    tA.data[j]=tA.data[j-1];
                }
                tA.data[0]=A.data[i];
                BigInt_Abdicate(tA.length,tA);
                if(BigInt_Judge(tA,B)!=-1)
                {
                    int k=0;
                    SeqList<int> tC,tD;
                    tC=tA;
                    BigInt_SUB(tC,B,tD);
                    while(tD.sign==1)
                    {
                        k++;
                        tC=tD;
                        BigInt_SUB(tC,B,tD);
                    }
                    C.data[i]=k;
                    tA=tC;
                    BigInt_Abdicate(tA.length,tA);
                }
                if(i==0)
                {
                    D=tA;
                }
```

```
        }
        BigInt_Abdicate(l1,C);
    }
}
```

3. 设计主函数

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include "SeqList.cpp"
using namespace std;
int  main( )
{
    string s1,s2;
    int l1,l2,r1[MaxSize],r2[MaxSize];
    cout<<"输入大整数A"<<endl;
    cin>>s1;
    cout<<"输入大整数B(做除数不为0)"<<endl;
    cin>>s2;
    l1=s1.length();
    l2=s2.length();
    for(int i=0;i<l1;i++)
        r1[l1-i-1]=s1[i]-'0';
    for(int i=0;i<l2;i++)
        r2[l2-i-1]=s2[i]-'0';
    SeqList<int> A(r1, l1);
    SeqList<int> B(r2, l2);
    SeqList<int> C,D,E,F,G;

    BigInt_ADD(A,B,C);
    BigInt_SUB(A,B,D);
    BigInt_MUL(A,B,E);
    BigInt_DIV(A,B,F,G);

    cout<<endl<<"A+B=";C.PrintList();
    cout<<endl<<"A-B=";D.PrintList();
    cout<<endl<<"A*B=";E.PrintList();
    cout<<endl<<"A/B整数部分为";F.PrintList();
    cout<<endl<<"A/B余数为"; G.PrintList();

    return 0;
}
```

六、运行测试

测试结果如图。

```
输入大整数A
1209372834029381
输入大整数B(做除数不为0)
23847293

A+B=1209372857876674

A-B=1209372810182088

A*B=28840268319339019315633

A/B整数部分为50713212

A/B余数为8494265
请按任意键继续. . .
```

七、总结与心得

大整数的运算相当于把我们熟知的加减乘除四则运算掰开揉碎打散重组，类似一种更加高阶的函数符号的重载。大整数的运算需要对基本的四则运算过程足够熟悉，不断进行调试，通过在纸上的演算和编程语言的运行，学会用程序语言表达数学思想。

在编码过程中，需要有良好的编码习惯，一个很重要的简化是对于顺序表的预处理，全部清零的那一个步骤。可以使得我们的退位与进位十分简单，达到了简化程序的效果。

这一次的实验相较于之前的实验更加复杂，对我们来说也是一次更加综合的运用和更大的挑战，在调试程序的过程中我自身的能力也得到了很大的提高。