浙江工业大学

数据结构课程设计

2022/2023(1)

课设题目 大整数运算

学生学号 202103150625

学生班级 软件工程（软件开发技术方向）01

任课教师 王松

提交日期 2022.12.8

计算机科学与技术学院

目录

大整数的运算 实验报告

一、 实验题目和要求

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

【基本要求】

1. 要求采用链表来实现大整数的存储和运算，**不允许使用标准模板类的链表类(list)和函数**。同时要求可以从键盘输入大整数，也可以文件输入大整数，大整数可以输出至显示器，也可以输出至文件。大整数的存储、运算和显示，可以同时支持二进制和十进制，但至少要支持十进制。大整数输出显示时，必须能清楚地表达出整数的位数。测试时，各种情况都需要测试，并附上测试截图；要求测试例子要比较详尽，各种极限情况也要考虑到，测试的输出信息要详细易懂，表明各个功能的执行正确；
2. 要求大整数的长度可以不受限制，即大整数的十进制位数不受限制，可以为十几位的整数，也可以为 500 多位的整数，甚至更长；大整数的运算和显示时，只需要考虑**正的大整数**。如果可能的话，请以秒为单位显示每次大整数运算的时间；
3. 要求采用类的设计思路，不允许出现类以外的函数定义，但允许友元函数。主函数中只能出现类的成员函数的调用，不允许出现对其它函数的调用。
4. 要求采用多文件方式：.h 文件存储类的声明，.cpp 文件存储类的实现，主函数 main 存储在另外一个单独的 cpp 文件中。如果采用类模板，则类的声明和实现都放在.h 文件中。
5. 不强制要求采用类模板，也不要求采用可视化窗口；要求源程序中有相应注释；
6. 要求采用 Visual C++ 6.0 及以上版本进行调试；

二、 实验开发环境

大整数的运算在 Visual Studio Code 平台下开发，操作系统：Windows 10。

硬件环境：

处理器：11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz

内存：40.0 GB

系统类型：64 位操作系统

三、 实验课题分析（主要的模块功能、流程图、算法原理、代码实现）

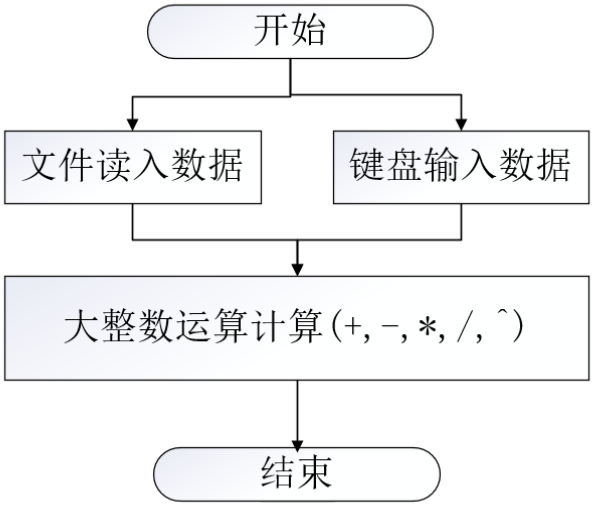
3.1 系统总体设计

1. 本程序实现任意长度的加法、减法、乘法、除法、指数运算。以用户和计算机的实现方式，即由计算机终端显示提示信息之后，由用户在键盘上输入任意长度的长整数并进行相应的运算，然后程序执行运算并将结果输出至显示屏中。
2. 在本程序中数字字符限定在 0—9，输入字符可任意长，通常长整数为 300~500 位的大整数。输入字符串过程以回车换行结束。
3. 利用双向循环链表实现长整数的存储，每一个节点存储一个整型变量。本程序仅仅

考虑均为正整数的情况。

4. 加、减、乘、除、指数运算是在取余操作下的加减乘除指数运算。
5. 既可以从键盘读入大整数进行相应操作也可以直接从文件中读入大整数进行相应操作。既可以将内存中的运行结果显示在屏幕中，也可以实现和硬盘中文件的连接将文件中内容显示在屏幕中。
6. 本实验重在运行准确性和运行效率。以秒为单位计算各个算法所运行的相应时间，并将程序运行时间显示在屏幕上。

3.2 系统功能设计



3.3 类的设计

1.节点类

```
class node {
    public:
        node();
        ~node();
        node( int n );

        int     value;
        node *pre;
        node *next;
};
```

2.链表类

```
class list{
        public:
            list();
```

```
        list(int);//每个节点都为 0 ,将链表初始化为长度为 n 的链表
        list(string);//以字符串构造
        ~list();

         void addATtail(int n);//在链表的末尾增加
        void addAThead(int n);//在链表的头部增加
         void init(string s);//初始化一个链表  string ->list of int
         void show();//显示 list 的信息
         void show() const ;

        string toString();//把链表中的 value 值拼接为一个 string
        string toString() const;
        list & operator=(const list &des);//等号重载,方便阅读代码
        bool operator>=(const list &des);//比较所代表数字的大小
        list sublist(int,int) const;//取出[int,int)的子串

        list add(const list &des) const;
        list sub(const list &des) const;
        list multi(const list &des) const;
        list multi(int) const;//只在除法中使用,int 乘链表所代表数字,返回一个链表
        list divide(const list &des) const;

        node *head;//头指针
        node *tail;//尾指针
        int length;//数字既链表的长度
};
```

3.大数类

```
class bigNumber{
    public:
        bigNumber();//无参构造方法,默认为 0
        bigNumber(string s);//以字符串初始化
        bigNumber(const bigNumber &b);
        ~bigNumber();

        friend ostream& operator<<(ostream&,const bigNumber&);//输出流重载
```

```
    friend istream& operator>>(istream&,bigNumber&); //输入流重载

    bigNumber int2bigNumber(int);//int 类型到 bigNumber 类型的转换
    void binary();//二进制输出
    void input_Binary();
    void showLength();//显示和长度

    bool    operator!=(int b) const;
    bool    operator==(int b) const;
    bigNumber &operator=(int b);
    bigNumber &operator=(const bigNumber & b);
    bool    operator==(const bigNumber &b) const;
    bool    operator!=(const bigNumber &b) const;
    bool    operator>(const bigNumber   &b) const;
    bool    operator<(const bigNumber   &b) const;

    bigNumber operator+(const bigNumber &b) const;
    bigNumber operator-(const bigNumber &b) const;
    bigNumber operator*(const bigNumber &b) const;
    bigNumber operator/(const bigNumber &b) const;
    bigNumber operator^(const bigNumber &b) const;//指数运算重载
    bigNumber operator^(int b) const;//指数运算重载
    bigNumber operator%(const bigNumber &b) const;
    bigNumber quickMod(const bigNumber &b ,const bigNumber &c);
  private:
    list number;
};
```

3.4 主要函数的设计

1.加法

从最低位开始，将两个加数对应位置上的数码相加，并判断是否达到或超过 10。如果达到，那么处理进位：将更高一位的结果上增加 1，当前位的结果减少 10 。

2.减法

从个位起逐位相减，遇到负的情况则向上一位借 。整体思路与加法完全一致。最后去零。

3.乘法

可以将其中一个乘数分解为它的所有数码，其中每个数码都是单精度数，将它们分别与另一个乘数相乘，再向左移动到各自的位置上相加即得答案。当然，最后也需要用与上例相同的方式处理进位。

4.除法

除法可以看作一个逐次减法的过程。为了减少冗余运算，我们提前得到被除数的长度与除数的长度，从被除数的长度减除数的长度的下标开始，从高位到低位来计算商。

5.取余

使用已完成的大整数运算实现：a-(a/b)*b　(a/b 为整除）。

6.幂次

计算 n 个 a 相乘，当 n 或 a 太大时速度会很慢，故采用了快速幂将时间缩短为 O(logn)。

四、 实验调试、测试

1.

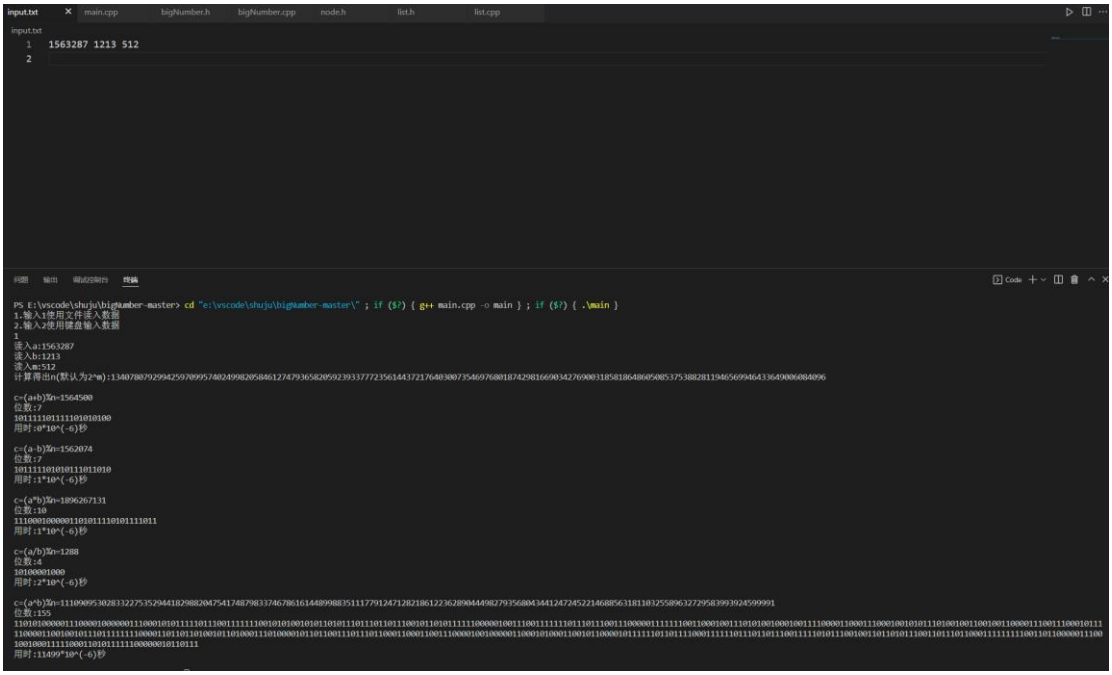
```
PS E:\vscode\shuju\bigNumber-master> cd "e:\vscode\shuju\bigNumber-master\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
1.输入1使用文件读入数据
2.输入2使用键盘输入数据
2
请输入a:
487843513322223
请输入b:
0
请输入m:
512
计算得出n(默认为2^m):13407807929942597099574024998205846127479365820592393377723561443721764030073546976801874298166903427690031858186486050853753882811946569946433649006084096

c=(a+b)%n=487843513322223
位数:15
110111011101100001110101101010001010010101011101111
用时:1*10^(-6)秒

c=(a-b)%n=487843513322223
位数:15
110111011101100001110101101010001010010101011101111
用时:1*10^(-6)秒

c=(a*b)%n=0
位数:1
0
用时:1*10^(-6)秒

除数不能为0
用时:1*10^(-6)秒

c=(a^b)%n=1
位数:1
1
用时:1*10^(-6)秒
```

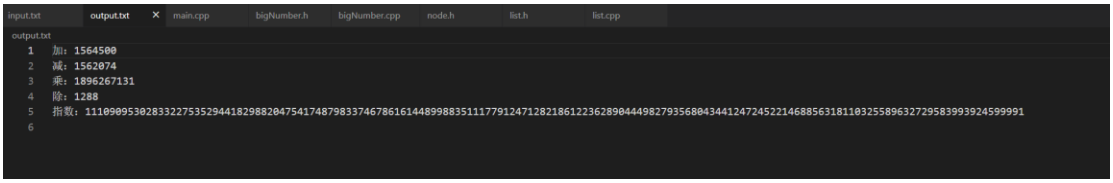
2.

```
PS E:\vscode\shuju\bigNumber-master> cd "e:\vscode\shuju\bigNumber-master\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
1.输入1使用文件读入数据
2.输入2使用键盘输入数据
2
请输入a:
1
请输入b:
487843513322223
请输入m:
512
计算得出n(默认为2^m):13407807929942597099574024998205846127479365820592393377723561443721764030073546976801874298166903427690031858186486050853753882811946569946433649006084096

c=(a+b)%n=487843513322224
位数:15
110111011101100001110101101010001010010101011110000
用时:0*10^(-6)秒

c=(a-b)%n=-487843513322222
为负数时位数不包括负号
位数:15
为负数时无二进制
用时:1*10^(-6)秒

c=(a*b)%n=487843513322223
位数:15
110111011101100001110101101010001010010101011101111
用时:1*10^(-6)秒

c=(a/b)%n=0
位数:1
0
用时:1*10^(-6)秒

c=(a^b)%n=1
位数:1
1
用时:102*10^(-6)秒
```

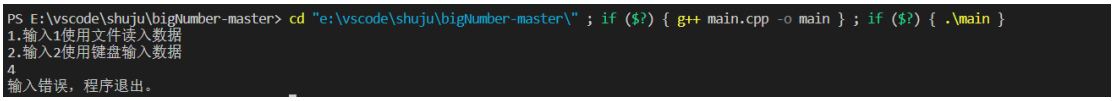
3.

4.





5.



五、 源代码

```
#include<iostream>

#include<string>

#include<time.h>

#include<fstream>
```

```
#include"bigNumber.cpp"
#include"node.cpp"
#include"list.cpp"
using namespace std;
int main(){
    ifstream input("input.txt");
    ofstream output("output.txt");
    clock_t start,end;
    bigNumber a,b,c,n("2"),m;
    int x;
    cout<<"1.输入 1 使用文件读入数据"<<endl;
    cout<<"2.输入 2 使用键盘输入数据"<<endl;
    cin>>x;
    if(x==1){
        input>>a;
        cout<<"读入 a:"<<a<<endl;
        input>>b;
        cout<<"读入 b:"<<b<<endl;
        input>>m;
        cout<<"读入 m:"<<m<<endl;
        n=n^m;
        cout<<"计算得出 n(默认为 2^m):"<<n<<endl<<endl;
    }else if(x==2){
        cout<<"请输入 a:"<<endl;
        cin>>a;
        cout<<"请输入 b:"<<endl;
        cin>>b;
        cout<<"请输入 m:"<<endl;
        cin>>m;
        n=n^m;
```

```
        cout<<"计算得出 n(默认为 2^m):"<<n<<endl<<endl;
    }else{
        cout<<"输入错误，程序退出。"<<endl;
        return 0;
    }

    start=clock();
    c=(a+b)%n;
    end=clock();
    cout<<"c=(a+b)%n="<<c<<endl;
    c.showLength();
    c.binary();
    cout<<"用时:"<<end-start<<"*10^(-6)秒"<<endl<<endl;
    output<<"加："<<c<<" "<<endl;

    start=clock();
    c=(a-b)%n;
    end=clock();
    if(a>b){
        cout<<"c=(a-b)%n="<<c<<endl;
        c.showLength();
        c.binary();
    }else{
        cout<<"c=(a-b)%n=-"<<c<<endl;
        cout<<"为负数时位数不包括负号"<<endl;
        c.showLength();
        cout<<"为负数时无二进制"<<endl;
    }
    cout<<"用时:"<<end-start<<"*10^(-6)秒"<<endl<<endl;
    output<<"减："<<c<<" "<<endl;
```

```
start=clock();
c=(a*b)%n;
end=clock();
cout<<"c=(a*b)%n="<<c<<endl;
c.showLength();
c.binary();
cout<<"用时:"<<end-start<<"*10^(-6)秒"<<endl<<endl;
output<<"乘：  "<<c<<" "<<endl;

start=clock();
c=(a/b)%n;
end=clock();
if(b!=bigNumber("0")){
    cout<<"c=(a/b)%n="<<c<<endl;
    c.showLength();
    c.binary();
}
cout<<"用时:"<<end-start<<"*10^(-6)秒"<<endl<<endl;
output<<"除：  "<<c<<" "<<endl;

start=clock();
c=a.quickMod(b,n);
end=clock();
cout<<"c=(a^b)%n="<<c<<endl;
c.showLength();
c.binary();
cout<<"用时:"<<end-start<<"*10^(-6)秒"<<endl<<endl;
output<<"指数：  "<<c<<" "<<endl;
```

```
    return 0;
}

#include"list.h"
#include"time.h"
#include<string>
using namespace std;
class bigNumber{
    public:
        bigNumber();//无参构造方法,默认为 0
        bigNumber(string s);//以字符串初始化
        bigNumber(const bigNumber &b);
        ~bigNumber();

        friend ostream& operator<<(ostream&,const bigNumber&);//输出流重载
        friend istream& operator>>(istream&,bigNumber&); //输入流重载

        bigNumber int2bigNumber(int);//int 类型到 bigNumber 类型的转换
        void binary();//二进制输出
        void input_Binary();
        void showLength();//显示和长度

        bool        operator!=(int b) const;
        bool        operator==(int b) const;
        bigNumber &operator=(int b);
        bigNumber &operator=(const bigNumber & b);
        bool        operator==(const bigNumber &b) const;
        bool        operator!=(const bigNumber &b) const;
        bool        operator>(const bigNumber    &b) const;
        bool        operator<(const bigNumber    &b) const;
```

```
        bigNumber operator+(const bigNumber &b) const;
        bigNumber operator-(const bigNumber &b) const;
        bigNumber operator*(const bigNumber &b) const;
        bigNumber operator/(const bigNumber &b) const;
        bigNumber operator^(const bigNumber &b) const;//指数运算重载
        bigNumber operator^(int b) const;//指数运算重载
        bigNumber operator%(const bigNumber &b) const;
        bigNumber quickMod(const bigNumber &b ,const bigNumber &c);
    private:
        list number;
};

#include "bigNumber.h"
bigNumber::bigNumber(){
    number.init("0");
}
bigNumber::~bigNumber(){
    this->number.~list();
}
bigNumber::bigNumber(string s){
    number.init(s);
}
bigNumber::bigNumber(const bigNumber &b){
    this->number.init(b.number.toString());
}

ostream& operator<<(ostream& output,const bigNumber& b){
    output<<b.number.toString();
    return output;
```

```
}
istream& operator>>(istream& input,bigNumber& b){
    string str;
    input>>str;
    b.number.init(str);
    return input;
}
bigNumber bigNumber::int2bigNumber(int n){
    string str;
    if(n==0){
        str="0";
    }else{
        while(n!=0){
            str+=(n%10)+'0';
            n/=10;
        }
    }
    bigNumber temp(str);
    return temp;
}
void bigNumber::input_Binary(){
    bigNumber two("2");
    string str;
    this->number.init("0");
    cout<<"请输入一个二进制数"<<endl;
    cin>>str;
    int l=str.length();
    for(int i=0;i<l;i++){
        if(str[i]=='1'){
            int zhishu=l-i-1;
```

```
                bigNumber temp=two^zhishu;
                (*this)=(*this)+temp;
            }
        }
    }
void bigNumber::binary(){
    string str;
    bigNumber two("2");
    bigNumber one("1");
    bigNumber temp;
    temp.number.init((*this).number.toString());
    while(temp>one){
        str=(temp%two).number.toString()+str;
        temp=temp/two;
    }
    if(temp==one)
        str="1"+str;
    if(temp==bigNumber("0"))
        str="0"+str;
    cout<<str<<endl;
}
void bigNumber::showLength(){
    int l=this->number.length;
    cout<<"位数:"<<l<<endl;
}

bigNumber &bigNumber::operator=(const bigNumber & des){
    this->number=des.number;
    return *this;
}
```

```
bigNumber &bigNumber::operator=(int b){
    bigNumber temp;
    temp=temp.int2bigNumber(b);
    this->number=temp.number;
    return *this;
}
bool bigNumber::operator!=(const bigNumber &b) const{
    if((*this)==b)
        return false;
    else
        return true;
}
bool bigNumber::operator!=(int b) const{
    bigNumber temp;
    temp=temp.int2bigNumber(b);
    return (*this)!=temp;
}
bool bigNumber::operator==(const bigNumber &b) const{
    string x=this->number.toString();
    string y=b.number.toString();
    if(x==y){
        return true;
    }else{
        return false;
    }
}
bool bigNumber::operator==(int b)const{
    bigNumber temp;
    temp=temp.int2bigNumber(b);
    return (*this)==temp;
```

```
}

bool bigNumber::operator>(const bigNumber & b) const{
    string x=this->number.toString();
    string y=b.number.toString();

    if(x.length()==y.length()){
        if(x==y){
            return false;
        }else{//zhu wei bi jiao
            int l=x.length();
            for(int i=0;i<l;i++){
                if(x[i]>y[i])return true;
                if(x[i]<y[i])return false;
            }
            return false;
        }
    }else if(x.length()>y.length()){
        return true;
    }else{
        return false;
    }
}
bool bigNumber::operator<(const bigNumber &b) const{
    if(*this==b){
        return false;
    }else if(*this>b){
        return false;
    }else{
        return true;
```

```
        }
    }



bigNumber bigNumber::operator+(const bigNumber & b) const{
        bigNumber temp((this->number.add(b.number)).toString());
        return temp;
}
bigNumber bigNumber::operator-(const bigNumber & b)const{
        bigNumber result;
        if((*this)>b||(*this)==b){//大于或等于就相减
                bigNumber temp((this->number.sub(b.number)).toString());
                result=temp;
        }else{
                bigNumber temp((b.number.sub(this->number)).toString());
                result=temp;
        }
        return result;
}
bigNumber    bigNumber::operator*(const bigNumber &b) const{
        bigNumber result(this->number.multi(b.number).toString());
        return result;
}
bigNumber bigNumber::operator/(const bigNumber &b) const{
        bigNumber result;
        bigNumber zero("0");
        bigNumber one("1");
        if(b==zero){
                cout<<"除数不能为 0"<<endl;
        }else if((*this)==b){
```

```
            result=1;
        }else if((*this)<b){
             result=zero;
        }else{
            bigNumber temp(this->number.divide(b.number).toString());
            result=temp;
        }
        return result;
    }
    bigNumber bigNumber::operator^(const bigNumber &b) const{
        bigNumber two("2");
        bigNumber one("1");
        bigNumber zero("0");
        bigNumber result;
        result.number.init(this->number.toString());//先将结果初始化为底数
        string moming=(*this).number.toString();
        if(b==0&&(*this)==0){
            cout<<"0 的 0 次没有意义"<<endl;
            result=0;
        }else if(b==0){
            result=one;
        }else{
            bigNumber aim=b;
            result=one;
            bigNumber second;
            bigNumber x;
            while(aim!=0){
                second=one;
                aim=aim-second;
                x=(*this);
```

```
                while(second<aim||second==aim){
                    aim=aim-second;
                    second=second*two;
                    x=x*x;
                }
                result=result*x;
            }
        }
        return result;
}
bigNumber bigNumber::operator^(int b) const{
        string str;
        if(b==0){
            str="0";
        }else{
            while(b!=0){
                str+=(b%10)+'0';
                b/=10;
            }
        }
        bigNumber temp;
        temp.number.init(str);
        return (*this)^temp;
}

bigNumber bigNumber::operator%(const bigNumber & b) const{
        return ((*this)- ((*this)/b)*b);
}
bigNumber bigNumber::quickMod(const bigNumber &b ,const bigNumber &c){
        bigNumber A,B,C;
```

```
    bigNumber zero("0");
    bigNumber two("2");
    A=(*this);
    B=b;
    C=c;
    bigNumber ans;
    ans=1;
    A=A%C;
    while(B>zero){
        if(B%two==1)
            ans=(ans*A)%C;
        B=B/two;
        A=(A*A)%C;
    }
    return ans;
}

#ifndef LIST_H
#define LIST_H
#include"node.h"
#include<string>
#include<iostream>
using namespace std;
class list{
    public:
        list();
        list(int);//每个节点都为 0 ,将链表初始化为长度为 n 的链表
        list(string);//以字符串构造
        ~list();
```

```
        void addATtail(int n);//在链表的末尾增加
        void addAThead(int n);//在链表的头部增加
        void init(string s);//初始化一个链表  string ->list of int
        void show();//显示 list 的信息
        void show() const ;

        string toString();//把链表中的 value 值拼接为一个 string
        string toString() const;
        list & operator=(const list &des);//等号重载,方便阅读代码
        bool operator>=(const list &des);//比较所代表数字的大小
        list sublist(int,int) const;//取出[int,int)的子串


        list add(const list &des) const;
        list sub(const list &des) const;
        list multi(const list &des) const;
        list multi(int) const;//只在除法中使用,int 乘链表所代表数字,返回一个链表
        list divide(const list &des) const;

        node *head;//头指针
        node *tail;//尾指针
        int length;//数字既链表的长度
};
#endif


#include"list.h"
list::list(){
    head=NULL;
    tail=NULL;
```

```
    length=0;
}
list::list(int n){
    string temp="";
    for(int i=0;i<n;i++){
        temp+='0';
    }
    this->init(temp);
}
list::list(string s){
    this->~list();
    int l=s.length();
    for(int i=0;i<l;i++){
        int n=s[i]-'0';
        if(n>=0&&n<=9){
            addATtail(s[i]-'0');
        }else{
            cout<<"构造字段包含非数字字符"<<endl;
        }
    }
}
list::~list(){
    while(length>0){
        node* temp=tail;
        tail=tail->pre;
        delete temp;
        length--;
    }
    head=NULL;//删除最后一个节点后将头指针置空
}
```

```
void list::addAThead(int n){
    node * temp=new node(n);
    if(length==0){
        head=temp;
        tail=temp;
        length++;
    }else{
        temp->next=head;
        head->pre=temp;
        head=temp;
        length++;
    }
}
void list::addATtail(int n){
    node *temp=new node(n);
    if(length==0){
        head=tail=temp;
    }else{
        temp->pre=tail;
        tail->next=temp;
        tail=temp;
    }
    length++;
}
void list::init(string s){
    this->~list();
    int l=s.length();
    for (int i=0;i<l;i++){
        int n=s[i]-'0';
```

```
            if(n>=0&&n<=9){
                addATtail(s[i]-'0');
            }else{
                cout<<"构造字段包含非数字字符"<<endl;
            }
        }
    }
    void list::show(){
        cout<<"头节点值"<<this->head->value<<endl;
        cout<<"头节点 pre"<<this->head->pre<<endl;
        cout<<"尾节点 next"<<this->tail->next<<endl;
        cout<<"list 的长度"<<this->length<<endl;
        cout<<"list 内容"<<this->toString()<<endl;
    }
    void list::show()const{
        cout<<"头节点值"<<this->head->value<<endl;
        cout<<"头节点 pre"<<this->head->pre<<endl;
        cout<<"尾节点 next"<<this->tail->next<<endl;
        cout<<"list 的长度"<<this->length<<endl;
        cout<<"list 内容"<<this->toString()<<endl;
    }

    string list::toString(){
        string str="";
        node *cur=head;
        while(cur!=NULL){
            str+=cur->value+'0';
            cur=cur->next;
        }
        return str;
```

```
}
string list::toString() const{
    string str="";
    node *cur=head;
    while(cur!=NULL){
        str+=cur->value+'0';
        cur=cur->next;
    }
    return str;
}

list & list::operator=(const list &des){
    this->init(des.toString());
    return *this;
}
bool list::operator>=(const list &des){
    string x=this->toString();
    string y=des.toString();
    if(x.length()==y.length()){
        if(x==y){//两个字符串相等
            return true;
        }else{//逐位比较
            int l=x.length();
            for(int i=0;i<l;i++){
                if(x[i]>y[i])
                    return true;
                if(x[i]<y[i])
                    return false;
            }
            return false;
```

```
            }
        }else if(x.length()>y.length()){
            return true;
        }else{
            return false;
        }
    }
    list list::sublist(int start,int end ) const{
        list temp;
        string str=this->toString();
        string sub_string=str.substr(start,end);
        temp.init(sub_string);
        return temp;
    }

    list list::add(const list &des) const{
        bool tiaoshi=false;
        node *t1=des.tail;//尾部指针
        node *t2=this->tail;
        int content=0;//每一位的计算结果
        int jingwei=0;//进位
        list result;//局部变量,存结果
        while(t1!=NULL||t2!=NULL||jingwei!=0){
            if(t1!=NULL){
                content+=t1->value;
                t1=t1->pre;
            }
            if(t2!=NULL){
                content+=t2->value;
                t2=t2->pre;
```

```
            }
            content+=jingwei;
            if(content>9){
                    jingwei=1;
                    result.addAThead(content-10);
            }else{
                jingwei=0;
                result.addAThead(content);
            }
            content=0;
        }
        if(tiaoshi){
            cout<<"list::add 结果"<<result.toString()<<endl;
        }
        return result;
    }

    list list::sub(const list &des) const{
        node *t1=des.tail;//尾部指针
        node *t2=this->tail;

        int content=0;//每一位的计算结果
        int jingwei=0;//进位
        list result;//结果
        while(t1!=NULL||t2!=NULL||jingwei!=0){
            if(t1!=NULL){
                content-=t1->value;
                t1=t1->pre;
            }
            if(t2!=NULL){
```

```
                content+=t2->value;
                t2=t2->pre;
            }
            content+=jingwei;
            if(content<0){
                jingwei=-1;
                result.addAThead(content+10);
            }else{
                jingwei=0;
                result.addAThead(content);
            }
            content=0;
        }
        //如果链表的一开始有大量的 0 要去除
        while(result.length>1&&result.head->value==0){
            node * tempNode=result.head;
            result.head=result.head->next;
            result.head->pre=NULL;
            delete tempNode;
            result.length=result.length-1;
        }
        return result;
    }

    list list::multi(const list &des) const{
        bool tiaoshi=false;
        //模拟手算乘法;
        //先判断长度,以减少计算次数
        list result;
        list x_long,x_short;//两个乘数,一长一短
```

```
if(this->length>des.length){
    x_long=*this;
    x_short=des;
}else{
    x_long=des;
    x_short=*this;
}
int l=x_short.length;
list temp_list;//用于储存每一次乘法的结果
node * x_long_pointer;
node * x_short_pointer=x_short.tail;
if(tiaoshi){
    cout<<"long"<<x_long.toString()<<endl;
    cout<<"short"<<x_short.toString()<<endl;
}
//短乘数每一位和长乘数相乘
for(int i=0;i<l;i++){
    int content=0;
    int jinwei=0;
    x_long_pointer=x_long.tail;
    while(x_long_pointer!=NULL||jinwei!=0){
        content+=jinwei;
        if(x_long_pointer!=NULL)
            content+=(x_short_pointer->value) * (x_long_pointer->value);
        jinwei=content/10;
        content=content%10;
        temp_list.addAThead(content);
        content=0;
        if(x_long_pointer!=NULL)
            x_long_pointer=x_long_pointer->pre;
```

```
            }
            //根据位数在末尾添加相应个数的 0
            for(int j=0;j<i;j++){
                temp_list.addATtail(0);
            }
            x_short_pointer=x_short_pointer->pre;
            result=result.add(temp_list);
            temp_list.~list();
        }
        //整理 result  防止出现乘 0 之后出现一大串 0
        while(result.length>1&&result.head->value==0){
            node * tempNode=result.head;
            result.head=result.head->next;
            result.head->pre=NULL;
            delete tempNode;
            result.length=result.length-1;
        }
        if(tiaoshi){
            result.show();
        }
        return result;
}

list list::multi(int n) const{
        if(n==0){
            list temp;
            temp.init("0");
            return this->multi(temp);
        }
        char char_n;
```

```
    string temp;
    while(n!=0){
        char_n=(n%10)+'0';
        temp=char_n+temp;
        n/=10;
    }
    list list_of_n;
    list_of_n.init(temp);
    return this->multi(list_of_n);
}

list list::divide(const list&des)const{
    bool tiaoshi=false;
    int l_of_long=this->length;//被除数
    int l_of_short=des.length;//除数

    list result;//结果
    node *divide_pointer=this->head;//指向被除数的某一位的一个指针
    for(int i=0;i<(l_of_short);i++){
        divide_pointer=divide_pointer->next;
    }
    int cishu=l_of_long-l_of_short+1;//要计算的位数(首位可能为 0)也就是要做循环的次数
    //todo 完成链表的=重载
    list jian_shu=this->sublist(0,l_of_short);//余数
    list bei_jian_shu;
    list yu_shu;
    int shang;
    for(int i=0;i<cishu;i++){
        shang=0;//商
        while((jian_shu.sub((des.multi(shang))))>=des){
```

```
            shang=shang+1;
        }

        result.addATtail(shang);
        bei_jian_shu=des.multi(shang);
        yu_shu = jian_shu.sub(bei_jian_shu);
        jian_shu=yu_shu;
        if(tiaoshi){
            cout<<endl<<"商"<<shang<<endl;
            cout<<"减数"<<jian_shu.toString()<<endl;
            cout<<"被减数"<<bei_jian_shu.toString()<<endl;
            cout<<"余数"<<yu_shu.toString()<<endl<<endl;
        }
        if(divide_pointer!=NULL){
            //需要判断 jian_shu 是否为 0
            //否则会出现开头为 0 的数字
            if(jian_shu.head->value==0){
                jian_shu.head->value=divide_pointer->value;
            }else{
                jian_shu.addATtail(divide_pointer->value);
            }
            divide_pointer=divide_pointer->next;
        }

    }
    //todo 去除掉首位的 0
    while(result.length>1&&result.head->value==0){
        node * tempNode=result.head;
        result.head=result.head->next;
        result.head->pre=NULL;
```

```
            delete tempNode;
            result.length=result.length-1;
        }
        if(tiaoshi){
            result.show();
        }
        return result;
    }



    #ifndef NODE_H
    #define NODE_H
    #include<iostream>
    using namespace std;
    class node{
        public:
            node();
            ~node();
            node(int n);

            int value;
            node *pre;
            node *next;
    };
    #endif



    #include"node.h"
    node::node(){
        pre=NULL;
```

```
        next=NULL;
    }
    node::~node(){}
    node::node(int n){
        value=n;
        pre=NULL;
        next=NULL;
    }
```