大数计算

实验报告

1551265 计 1 张伯阳

2017/6/1

1. 大数运算程序的设计与实现

1.1.操作方法及要求

【基本作业要求:】

1、定义一个 bigint 类，用来表示一个超过 int 类型可表示范围的大数，可以进行运算并能得到正确的结果

2、要求完成的运算为"加、减、乘、除、模、正号、负号、赋值、复合赋值、自增、自减、比较、数组、函数调用"

3、通过 C++的运算符重载方式实现，操作数仅限于整数，运算结果也是整数

4、数制为十进制有符号数（支持正负数），要求能进行正确运算的大数不少于 1000000 位(十进制)

5、实现的中间过程中允许借用编译器中超过 4 字节的大整数定义（例如：long long int 等）

6、测试的 main 函数采用如下形式：

```
bigint fun(bigint x)
{
   ...
}
int main()
{
   bigint a, b;
    cin >> a >> b;
    cout << a+b << endl;
    cout << a-b << endl;
    cout << a*b << endl;
    cout << a/b << endl;
cout << a%b << endl;
cout << (a+=b) << endl;
cout << (a=b) << endl;
cout << ++a << ' ' << b-- << endl;
cout << -a << ' ' << a << endl;
cout << (a<b) << endl;
fun(a+b);
    return 0;
}


bigint fun(bigint x[])
{
   ...
}
int main()
{
   bigint a[10], b[5];
    cin >> a[2] >> b[3];//要支持循环读入数组
```

```
        cout << a[2]+b[3] << endl;
        cout << a[0]-b[4] << endl;
        cout << a[7]*b[1] << endl;
        cout << a[4]/b[0] << endl;
    cout << a[2]%b[3] << endl;
    cout << (a[0]+=b[4]) << endl;
    cout << (a[8]=b[1]) << endl;
    cout << ++a[3] << ' ' << b[2]-- << endl;
    cout << -a[0] << ' ' << a[0] << endl;
    cout << (a[6]<b[0]) << endl;
    fun(a);
        return 0;
    }
```

因为测试数据很大，建议采用输入输出重定向方式读入，每个 bigint 间用回车/空格分隔开

● 建议写一个辅助程序，

用于生成输入重定向文件的数字

（注意：即使一个 bigint 有 100 万位，中间也不应该有空格或换行，否则会当做两个数处理）

● 输出重定向文件查看时，肉眼观察不可能，建议大家使用相同输入数据和 main 函数进行测试（欢迎大家提供），然后各自上传输出结果文件，在 cmd 下用 comp 命令比较不同同学的输出结果文件来观察正确与否

7、存储 bigint 时，不能采用一次申请全部空间的方式进行，但允许适度浪费（10KB 以内）

（例：先申请 10KB 空间，存储 10240 位数字以内的 bigint，如果超过了，再扩大...）

8、提交的作业由四个文件组成，各文件的说明如下：

90-b5.h	：bigint 类的定义、需要的全局定义、函数声明等
90-b5.cpp	：bigint 类的实现
90-b5-main.cpp	：测试用例，要求不同同学的测试用例可互换并编译通过

2. 整体设计思路

```
class BigInt{
private:
    vector<char> a;//大数储存形式
    int len;//大数长度
    bool neg;//负数
public:
    BigInt();//构造函数
    BigInt(const int);//将int型转换为大数
    ~BigInt();//析构函数
    BigInt &operator=(const BigInt &);//赋值函数

    friend istream& operator >> (istream&, BigInt&);//输入函数
    friend ostream& operator << (ostream&, BigInt&);//输出函数

    BigInt operator+(const BigInt &) const;//加
    BigInt operator-(const BigInt &) const;//减
    BigInt operator*(const BigInt &) const;//乘
    BigInt operator/(const BigInt &) const;//除
    BigInt operator%(const BigInt &) const;//取模
    BigInt operator ++();//前置自增
    BigInt operator ++(int);//后置自增
    BigInt operator --();//前置自减
    BigInt operator --(int);//后置自减
    BigInt operator +();//正号
    BigInt operator -();//负号
    BigInt operator +=(const BigInt &);//加等
    BigInt operator -=(const BigInt &);//减等
    BigInt operator *=(const BigInt &);//乘等
    BigInt operator /=(const BigInt &);//除等
    BigInt operator %=(const BigInt &);//模等
    bool operator>(const BigInt & T) const;//大于比较
    bool operator>=(const BigInt & T) const;//大于等于比较
    bool operator<(const BigInt & T) const;//小于比较
    bool operator<=(const BigInt & T) const;//小于等于比较
    bool operator==(const BigInt & T) const;//等于比较
};
```

3. 主要功能的实现

输入两个大数,计算出其各种运算的值并输出

运算包括加、减、乘、除、模、正号、负号、赋值、复合赋值、自增、自减、比较、数组、函数调用

实现构造函数可以将BigInt初始化或将int型数转换为BigInt型

使用vector容器储存BigInt的数值域,将每位的储存类型设置为char,相比于short和int更节省空间.用len储存大数长度,neg储存负号.将负号与数据域分离,避免了很多计算中不必要的问题.

重载成员函数:

赋值:只需要将BigInt中private的三个成员依次赋值即可.

输入:先将输入的字符串用string保存,用.length()确定其长度,并将容器的长度调整为需要的长度,将string的每一位减去’0’按位赋值给a,即此vector中char保存的内容即为实际值.若第一位为负号,则neg置1.

输出:若BigInt的neg为1,则先输出负号,下来直接将数据域转换为int型输出.

加:先用异或判断负号位,若结果为1,直接转到减法进行计算.否则进行加法计算.
通过加数符号确定最终结果的符号,将加数和被加数符号去掉进行计算,防止符号产生影响.
将两数的位数均补齐到两数中最大位数加1,防止进位导致vector越界, 然后进行模仿人手动乘法计算,最后判断最高位是否为0,若不为零,则len+1.

减: 先用异或判断负号位,若结果为1,直接转到加法进行计算.否则进行减法计算.
通过两数的大小比较确定最终结果的符号,后将被减数和减数进行调整,使被减数和减数都为正数且被减数大于减数,防止产生符号影响.
若为两数符号相同的计算,则不可能发生结果位数比最大位数多的情况,所以将两数的位数均补齐到两数中最大位数即可,后进行模仿人手动减法计算,最后判断高位有多少0,全部在len中进行消除.

乘:通过两数符号异或确定结果符号.设结果ret为BigInt类型,ret位数置为两数len之和保证不越界,后进行模仿人手动乘法计算,最后判断高位的0的个数,逐个在ret的len中进行消除.

除: 通过两数符号异或确定结果符号.本次实现因时间有限不准备使用分组的方法,直接使用了多次相减的方法,这种方法实现起来比较清晰明了,代码写起来也很容易,但是缺点也很明显:计算很慢.使用BigInt型的ret储存最后商的结果,每次讲被除数减去除数并将ret自增1.

自增/减:分为前/后置两种,不过实现起来都较为简单,只需注意最后输出的结果.

正/负号:直接输出原值或输出改变(*this).neg即可.

复合赋值:只需在原操作基础上将操作结果赋值给(*this)即可

模:简便方法应该用分段取模或快速幂,在这里为了简便起见,直接用原数减去商和除数乘积得到最后模的结果.考虑到两负数的模用这种方法求出来是个负数,所以要再加上一个限制条件.

比较:分大/小于,等于,大/小于等于比较5种,以大于比较进行说明,先比较正负号,若符号不同则可以直接判断大小.若符号相同,len不同,也可以直接判断,否则从高位到低位逐位比较大小.

4. 调试过程碰到的问题

碰到的问题主要有下:

1. 用string字符串存储数据域出现错误

最初想法是使用string存储,其可以自动延长长度的特性好像很合适.但是发现其不如

char数组容易控制,不易清空和初始化,不同位数的数进行操作时,每次都会弹窗提示string越界,因为string是一个储存字符串且自动化较强的数组,虽然逻辑都没什么问题,但是因对其内部结构不清楚,所以最后放弃.

2. 用char数组储存数据域出现错误

   第二个想法是用char数组进行储存,即用传统的数组进行储存,至于使用short还是char只是对内存的占用不同了.但是在动态改变数组大小时又出现了问题.在类中用calloc改变(*this)中的数组大小,出现了无法修改左值的问题,在网上搜索了很多也没有完美解决的办法,我也不知道是calloc在成员函数中不能使用还是怎么的.虽然我可以用在成员函数中在重新声明一个BigInt类型变量进行操作,得到最后计算结果,但是如果到了需要改变原值的时候,比如乘等于或赋值给原数时候就会出现这个无法避免的问题:我可以很容易改变private的值,但是却很难改变private中数组的大小(也许是我没有找到).反正这种方案就差一点就实现了,但是最后还是因为不能动态改变数组大小而放弃.

3. 用vector容器储存数据域

   第三个想法是用一个可以任意添加删除元素的容器实现操作.相比之下vector实现起来还是容易了很多的,可以直接很方便的在指定位置insert指定个数的初始化元素,在类中兼容性也比较好,不会出现无法修改的现象.需要改变的前期的函数也很少,运行起来稳定性还是不错的,只有之前少量的几个bug,修改完就通过了我的测试样例.但是因为时间紧张,我没有再进行较多的数据测试.

5. 心得体会

从这次作业我得到的主要教训有下:

1. 比较抽象的代码要有一些必要的注释

   之前一天写的代码后一天看起来已经有些困难了,更不要说过很久再检查.

   里面的步骤实在非常抽象,如果没有注释的提示需要好久的代码重读才能看出到底是在实现什么功能.每个函数内的每块代码也应该有相应的注释来解释和说明,否则后期检查逻辑错误时会非常困难.对特定的变量最好在变量名就将其标准化,这样不需要很多的注释也很容易理解.

2. 代码复用

   每个函数相对完成的功能相对独立,基本只在传参方面有影响.函数内部进行了不少的代码复用,如优化循环优化判断,函数的复用,使代码量有大幅度减少.

这次的大作业的完成,我用了一天多点的时间,主要函数的实现用的时间还是比较多的,又因为是边debug边完成代码,效率高了一些.最后关于动态申请内存,用进行了很多尝试和修改.

6. 附件：源程序

```
/* 1551265 计1 张伯阳 */
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
#include<string>
#include<iomanip>
#include<vector>
#include<algorithm>
using namespace std;

#define MAXN 9
#define LEN 1024

class BigInt {
private:
    vector<char> a;//大数储存形式
    int len;//大数长度
    bool neg;//负数
public:
    BigInt();//构造函数
    BigInt(const int);//int转大数
    ~BigInt();//析构
    BigInt &operator=(const BigInt &);//赋值

    friend istream& operator >> (istream&, BigInt&);//输入
    friend ostream& operator << (ostream&, BigInt&);//输出

    BigInt operator+(const BigInt &) const;//加
    BigInt operator-(const BigInt &) const;//减
    BigInt operator*(const BigInt &) const;//乘
    BigInt operator/(const BigInt &) const;//除
    BigInt operator%(const BigInt &) const;//取模
    BigInt operator ++();//前置自增
    BigInt operator ++(int);//后置自增
    BigInt operator --();//前置自减
    BigInt operator --(int);//后置自减
    BigInt operator +();//正号
    BigInt operator -();//负号
    BigInt operator +=(const BigInt &);//加等
    BigInt operator -=(const BigInt &);//减等
    BigInt operator *=(const BigInt &);//乘等
    BigInt operator /=(const BigInt &);//除等
    BigInt operator %=(const BigInt &);//模等
    bool operator>(const BigInt & T) const;//大于比较
    bool operator>=(const BigInt & T) const;//大于等于比较
    bool operator<(const BigInt & T) const;//小于比较
    bool operator<=(const BigInt & T) const;//小于等于比较
    bool operator==(const BigInt & T) const;//等于比较
};

/* 1551265 计1 张伯阳 */
#include "90-b5.h"
```

```
BigInt::BigInt()
{
    len = 1;
    a.push_back(0);
    neg = 0;
}
BigInt::BigInt(const int b)//int转大数
{
    int c, d = b;
    len = 0;
    neg = b < 0;
    if (neg)
        d = -d;
    a.insert(a.end(), LEN, 0);
    while (d > MAXN)
    {
        c = d % (MAXN + 1);
        d = d / (MAXN + 1);
        a[len++] = c;
    }
    a[len++] = d;
}
BigInt::~BigInt()
{
    vector<char>(a).swap(a);
}
BigInt & BigInt::operator=(const BigInt & n)//赋值
{
    (*this).a = n.a;
    (*this).len = n.len;
    (*this).neg = n.neg;
    return *this;
}
istream& operator >> (istream & in, BigInt & b)//输入
{
    string str;
    in >> str;
    int len = str.length();
    b.a.insert(b.a.end(), len, 0);
    if (str[0] == '-')
    {
        b.neg = 1;
        for (int i = 0; i < len - 1; i++)
            b.a[i] = str[len - i - 1] - '0';
        b.len = len - 1;
    }
    else
    {
        b.neg = 0;
        for (int i = 0; i < len; i++)
            b.a[i] = str[len - i - 1] - '0';
        b.len = len;
    }
    return in;
```

装 订 线

```
}
ostream& operator<<(ostream& out, BigInt& b)//输出
{
    if (b.neg)
        cout << "-";
    for (int i = b.len - 1; i >= 0; i--)
        cout << int(b.a[i]);
    return out;
}

BigInt BigInt::operator+(const BigInt & T) const//加
{
    BigInt t(*this),TT(T);
    int big,flag=0;
    if (t.neg^T.neg && t.neg)
    {
        t.neg = 0;
        return TT - t;
    }
    else if (t.neg^T.neg)
    {
        TT.neg = 0;
        return t - TT;
    }
    big = TT.len > len ? TT.len : len;
    if (big > t.len)
    {
        t.a.insert(t.a.end(), big - t.len + 1, 0);
        TT.a.push_back(0);
    }
    else
    {
        TT.a.insert(TT.a.end(), big - TT.len + 1, 0);
        t.a.push_back(0);
    }
    for (int i = 0; i < big; i++)
    {
        t.a[i] += TT.a[i];
        if (t.a[i] > MAXN)
        {
            t.a[i + 1]++;
            t.a[i] -= MAXN + 1;
        }
    }
    if (t.a[big] != 0)
        t.len = big + 1;
    else
        t.len = big;
    t.neg = neg&&T.neg;
    if (t.len == 1&&t.a[0]==0)
        t.neg = 0;
    return t;
}
BigInt BigInt::operator-(const BigInt & T) const//减
```

装 订 线

```
{
    int i, j, big;
    bool flag = !((*this) >= T);
    BigInt t1= (*this), t2 = T;
    if (t1.neg^t2.neg && t1.neg)
    {
        t2.neg = 1;
        return t1 + t2;
    }
    else if (t1.neg^t2.neg)
    {
        t2.neg = 0;
        return t1 + t2;
    }
    t1.neg = t2.neg = 0;
    if (!(t1 >= t2))
    {
        BigInt t3;
        t3 = t1;
        t1 = t2;
        t2 = t3;
    }
    t2.a.insert(t2.a.end(), len - t2.len + 1, 0);
    t1.a.push_back(0);
    big = t1.len;
    for (i = 0; i < big; i++)
    {
        if (t1.a[i] < t2.a[i])
        {
            j = i + 1;
            while (t1.a[j] == 0)
                j++;
            t1.a[j--]--;
            while (j > i)
                t1.a[j--] += MAXN;
            t1.a[i] += MAXN + 1 - t2.a[i];
        }
        else
            t1.a[i] -= t2.a[i];
    }
    t1.len = big;
    while (!t1.a[t1.len - 1] && t1.len > 1)
    {
        t1.len--;
        big--;
    }
    t1.neg = flag;
    if (t1.len == 1 && t1.a[0] == 0)
        t1.neg = 0;
    return t1;
}

BigInt BigInt::operator*(const BigInt & T) const//乘
{
```

```
    BigInt ret;
    ret.a.insert(ret.a.end(), len + T.len, 0);
    int i, j, up, temp, temp1;
    for (i = 0; i < len; i++)
    {
        up = 0;
        for (j = 0; j < T.len; j++)
        {
            temp = a[i] * T.a[j] + ret.a[i + j] + up;
            if (temp > MAXN)
            {
                temp1 = temp % (MAXN + 1);
                up = temp / (MAXN + 1);
                ret.a[i + j] = temp1;
            }
            else
            {
                up = 0;
                ret.a[i + j] = temp;
            }
        }
        if (up != 0)
            ret.a[i + j] = up;
    }
    ret.len = i + j;
    while (ret.a[ret.len - 1] == 0 && ret.len > 1)
        ret.len--;
    ret.neg = neg^T.neg;
    if (ret.len == 1 && ret.a[0] == 0)
        ret.neg = 0;
    return ret;
}
BigInt BigInt::operator/(const BigInt & T) const//除
{
    BigInt t = (*this), ret = 0, TT = T;
    int big;
    bool flag= neg^T.neg;
    t.neg = 0, TT.neg = 0;
    big = TT.len > len ? TT.len : len;
    if (big > TT.len)
        TT.a.insert(TT.a.end(), big - TT.len, 0);
    else if (big > t.len)
        t.a.insert(t.a.end(), big - t.len, 0);
    ret.a.insert(ret.a.end(), big, 0);
    if (t == 0)
        return 0;
    if (TT == 0)
    {
        cout << "error!" << endl;
        return 0;
    }
    while (t > TT || t == TT)
    {
        t = t - TT;
```

```
            ret = ret + 1;
        }
        ret.neg = flag;
        if (ret.len == 1 && ret.a[0] == 0)
            ret.neg = 0;
        return ret;
    }
    BigInt BigInt::operator++()//前置自增
    {
        (*this) = (*this) + 1;
        return (*this);
    }

    BigInt BigInt::operator++(int)//后置自增
    {
        (*this) = (*this) + 1;
        return (*this) - 1;
    }

    BigInt BigInt::operator--()//前置自减
    {
        (*this) = (*this) - 1;
        return (*this);
    }

    BigInt BigInt::operator--(int)//后置自减
    {
        (*this) = (*this) - 1;
        return (*this) + 1;
    }

    BigInt BigInt::operator+()//正号
    {
        return (*this);
    }

    BigInt BigInt::operator-()//负号
    {
        BigInt c = (*this);
        c.neg = !c.neg;
        return c;
    }

    BigInt BigInt::operator +=(const BigInt & b)//加等
    {
        (*this) = (*this) + b;
        return (*this);
    }
    BigInt BigInt::operator -=(const BigInt & b)//减等
    {
        (*this) = (*this) - b;
        return (*this);
    }
    BigInt BigInt::operator *=(const BigInt & b)//乘等
```

```
{
    (*this) = (*this) * b;
    return (*this);
}
BigInt BigInt::operator /=(const BigInt & b)//除等
{
    (*this) = (*this) / b;
    return (*this);
}
BigInt BigInt::operator %=(const BigInt & b)//模等
{
    (*this) = (*this) % b;
    return (*this);
}

BigInt BigInt::operator %(const BigInt & b) const//取模
{
    BigInt c = (*this) - (((*this) / b)*b);
    if (c < 0)
        c += b;
    return c;
}
bool BigInt::operator>(const BigInt & T) const//大于比较
{
    int ln;
    if (neg && !T.neg)
        return false;
    if (!neg&&T.neg)
        return true;
    if (len > T.len&&!neg)
        return true;
    if (len > T.len&& neg)
        return false;
    if (len == T.len && !neg)
    {
        ln = len - 1;
        while (ln >= 0&& a[ln] == T.a[ln] )
            ln--;
        if (ln >= 0 && a[ln] > T.a[ln])
            return true;
        else
            return false;
    }
    else if (len == T.len&&neg)
    {
        ln = len - 1;
        while (ln >= 0&&a[ln] == T.a[ln]  )
            ln--;
        if (ln >= 0 && a[ln] < T.a[ln])
            return true;
        else
            return false;
    }
    else if (!neg)
```

```
        return false;
    else
        return true;
}
bool BigInt::operator<(const BigInt & T) const//小于比较
{
    int ln;
    if (neg && !T.neg)
        return true;
    if (!neg&&T.neg)
        return false;
    if (len > T.len && !neg)
        return false;
    if (len > T.len&& neg)
        return true;
    if (len == T.len && !neg)
    {
        ln = len - 1;
        while (ln >= 0&&a[ln] == T.a[ln]  )
            ln--;
        if (ln >= 0 && a[ln] < T.a[ln])
            return true;
        else
            return false;
    }
    else if (len == T.len&&neg)
    {
        ln = len - 1;
        while (ln >= 0&&a[ln] == T.a[ln]  )
            ln--;
        if (ln >= 0 && a[ln] > T.a[ln])
            return true;
        else
            return false;
    }
    else if (!neg)
        return true;
    else
        return false;
}
bool BigInt::operator==(const BigInt & T) const//等于比较
{
    int ln;
    if (len != T.len)
        return false;
    if (neg != T.neg)
        return false;
    else
    {
        ln = len - 1;
        while (ln > 0 && a[ln] == T.a[ln])
            ln--;
        if (!ln && a[ln] == T.a[ln])
            return true;
```

```
        else
            return false;
    }
}
bool BigInt::operator<=(const BigInt & T) const//小于等于比较
{
    if ((*this) < T || (*this) == T)
        return true;
    else
        return false;
}
bool BigInt::operator>=(const BigInt & T) const//大于等于比较
{
    if ((*this) > T || (*this) == T)
        return true;
    else
        return false;
}
int main()
{
    BigInt a, b;
    cin >> a >> b;
    cout << a + b << endl;
    cout << a - b << endl;
    cout << a*b << endl;
    cout << a / b << endl;
    cout << a%b << endl;
    cout << (a += b) << endl;
    cout << (a = b) << endl;
    cout << ++a << ' ' << b-- << endl;
    cout << -a << ' ' << a << endl;
    cout << (a < b) << endl;
    //fun(a + b);
    return 0;
}
```

装 订 线