同济大学 实验报告

装

订

线

大数运算bigint

---C++程序设计实验报告

姓名：童佳燕

学号：1551445

班级：计算机科学与技术1班

指导教师：沈坚

完成日期：2017年6月6日

1. 实验题目：大数运算bigint

1.1 实验要求

（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）要求能进行正确运算的大数不少于1000000位（十进制）

装

1.2 程序设计要求

（1）独立功能使用独立的函数实现，严格按照格式编写，强调注释以及函数，变量的命名可视化。

订

（2）不允许使用goto语句和string类，以及全局变量，允许使用string

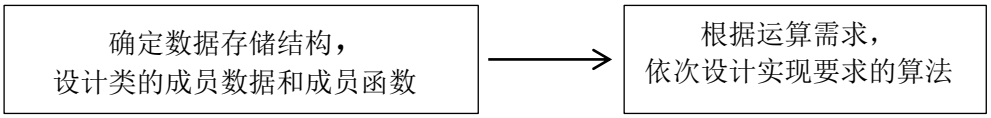
（3）涉及申请动态空间，要求自行释放

线

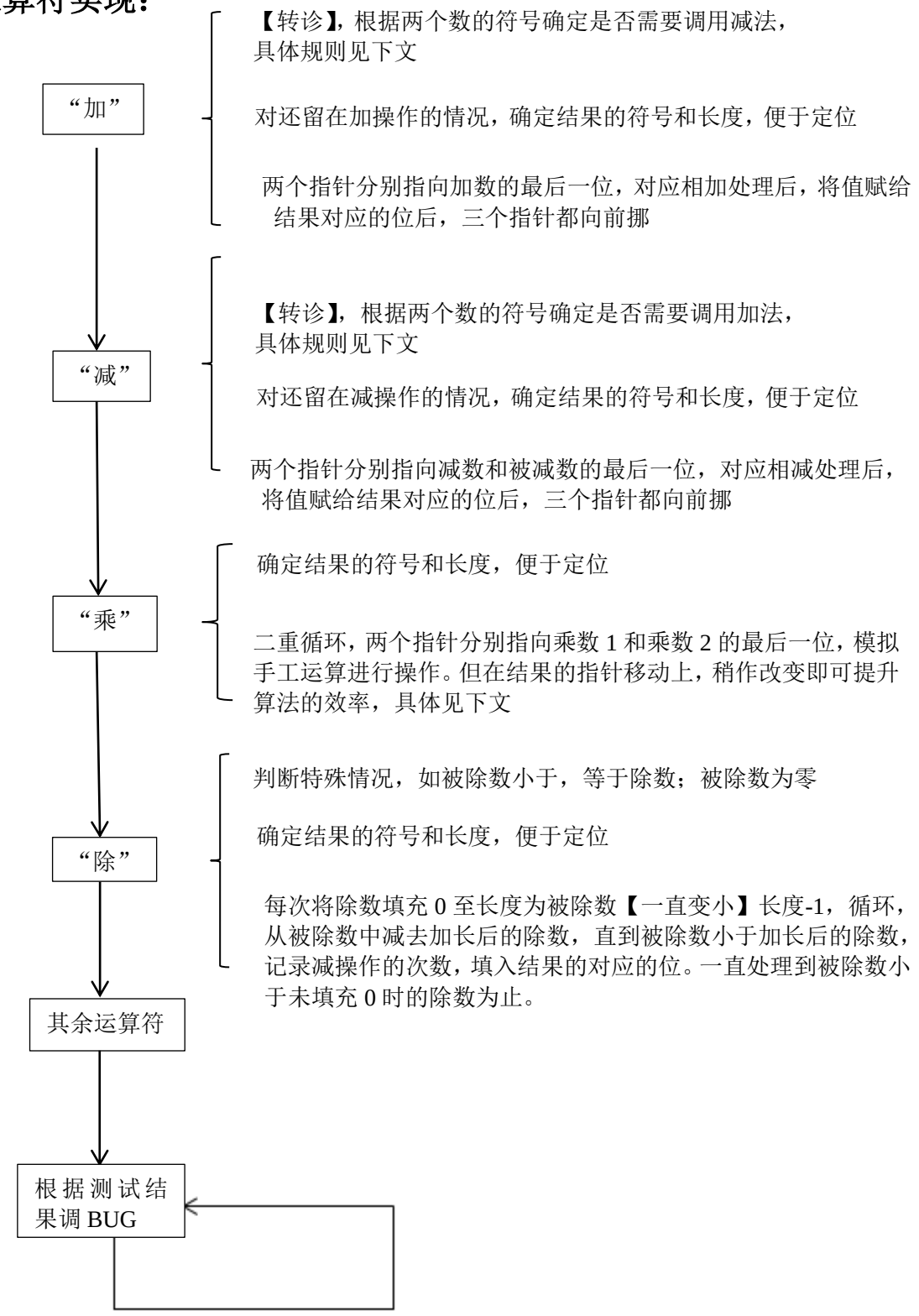
（4）存储bigint 时不允许一次申请全部空间的方式进行，但允许适度浪费,不超过10KB

2. 整体设计思路

大致框架如下图：



主要运算符实现：



其余操作如模，考虑到先除再乘最后减的效率太低，于是采用的是类似除的操作，记录循环结束后的被除数，即模的结果；前缀/后缀++/--则是加法减法的简化版，处理好返回值即可；这些操作都比较简单，具体请看源代码的实现。

3.主要功能的实现

3.1 加法

一开始将指针pa,pb,pr分别指向加数1，加数2以及“和”的最后一位，如图。相加对象是pa，pb指向的字符减去‘0’转化成int后的值相加，以及carry的值。相加后的结果需要考虑大于十时，填入pr指向的地址的值应是相加后结果模十之后的结果，同时进位carry等于相加后结果除以十的结果；

需要注意的问题【部分与减法操作时相同，不赘述】,

（1）bigint中存储bigint内容的是string型，其中的“数字”是字符型的数字，因此在做/,%等操作是，需要减去‘0’，转换成int型。

（2）在确定结果长度是，由于x位的数与y位的数相加时，结果的位数为x+y-1或者x+y，而在函数一开始判断长度时为了避免空间超，是设置成为x+y的，所以在最后结果出来后需要进行判断，首位是否为‘0’，如果是需要将其消掉，利用string自带的函数assign.

（3）除了结果的长度需要注意，由于执行过程中pa，pb一直在向前移动，需要注意会存在其中一个加数的长度不够长，因此需要加入判断，如果到达加数的“头”后，指针不在前移，操作中变为另一个加数对应的位与carry进行相加。

（4）并不是所有都适合用加法，为了简便操作，函数一开始就对两个加数进行判断，确定是否需要转诊到减函数中。对所有正负情况进行分析，如下：

正整数+正整数，不变

正整数-正整数，不变

正整数-负整数，转换成 正整数+正整数；

正整数+负整数，转换成 正整数-正整数；

负整数1+正整数2，转换成 正整数2-正整数1

负整数1-负整数2，转换成 正整数2-正整数1

负整数-正整数，转换成 （-1）正整数+正整数

负整数1+负整数2，转换成 （-1）正整数+正整数

其中黑色字体需要在减法函数中完成，红色在加法函数中完成，底色标灰的是需要转诊的。在转诊的过程中，需要将符号位负的数转成正，传入新的函数中。

装

3.2减法

同加法一样，首先需要对进来的数进行正负判断，根据具体情况判断是否需要“转诊”。对于不需要转诊的情况，一开始将指针pa,pb,pr分别指向被减数，减数以及“差”的最后一位。操作对象是*pa，*pb指向的字符相减后，再减去borrow的值。相减后的结果需要考虑小于0时，填入pr指向的地址的值应是相减后结果加十之后的结果，同时借位carry等于-1；而如相减后结果大于等于0，则borrow为0。

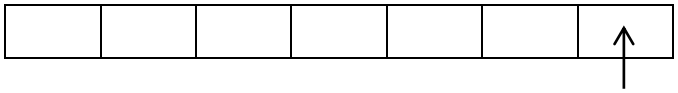
需要注意的问题，除上述加法操作数所列，还需要注意，为了尽量避免后续的讨论，在一开始设置指针指向时，将pa指向较大的数（不考虑符号，都处理成正整数的加减），这同加法不同，减法需要提前进行判断。

订

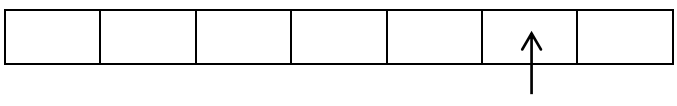
3.3乘法

利用二重循环，两个指针分别指向乘数 1 和乘数 2 的最后一位，模拟手工运算进行操作。但在结果的指针移动上，稍作改变即可提升算法的效率。具体实现见源码以及注释：

（1）第一次外循环，pr 指针指向最后一位，如图所示；随着内循环的进行，指针向前移动，并在所指向的地址中填入相乘加上 carry 后，模十的结果，同时记录进位 carry，加在下一轮内循环的相乘后的结果上。



（2）第二次外循环，pr指针移到如图所示的位置，再随着内循环的进行向前移动，向所指向的地址中填入　相乘加上carry，加上现有值后，模十的结果，同时记录进位carry，加在下一轮内循环的相乘后的结果上。



线

（3）…….(重复（2）操作至结束)

为了使操作统一，在result初始化时，初始化为相应长度的“00000……”,这样就可以循环做（2）的操作。

3.4 除法

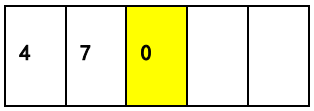
为了解释清楚具体操作，篇幅问题，用较短的数进行举例说明：

【564994/12】:

（1） 12 → 12000 → 564994-12000=552994 → 552994-12000=540994 →…12996-12000-994

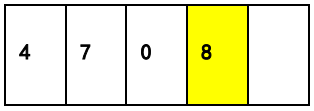
① ② …… ㊼

（2）12 → 1200 → 994<1200 continue;

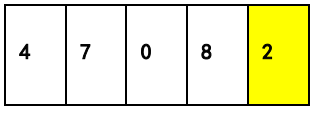


(3)12 → 120 → 994-120=874→874-120=754…… 154-120=34

① ② ⑧



(4)12→34-12=22→22-12=10



4. 调试过程碰到的问题

4.1 出现乱码

这是一开始遇到的bug,1+1后会出现乱码;实际上是由于忘记了bigint中“数字”是字符型，在做/,%时，需要先减去’0’转换成int型。

4.2 （-4）++结果是-5

由于操作时是将符号隔离开操作，最后在附上符号，所以在做前后缀++/--时忽略了这个点。不过这个错误还是很容易找到源头的。最终在前后缀++/--时加入了参与运算的对象的符号判断，采用如+/-中的转诊机制。

装

5. 经验总结&经验教训

(1)string真好用！但是必须要面临的是，不是所有语言都有这个玩意儿，遇到语言不支持string时，要有能力去自己构造这样一个工具。

（2）写完程序后才意识到自己没有做到静态函数，以及const参数的设置。原因之一是，那部分没好好听课，不是很熟悉，还需要好好学习课件，同时多练习才可以熟练掌握。

订

6. 附件：源程序

线

```
#define P 1//positive 正数
#define N 0//negative f负数
class bigint
{
private:
    char sign;//正负
    unsigned int len;//位数 strlen(content)
    string content;//存储内容,不包括sign
public:
    bigint();//初始化，sign=P,len=2,存放sign
    bigint(const string str);//初始化
    bigint operator+(bigint &b);
    bigint operator-(bigint &b);
    bigint operator*(bigint &b);
    bigint operator/(bigint &b);
    bigint operator%(bigint &b);
    bigint operator[](int i);
    bigint operator+();//正号
    bigint operator-();//负号
    bigint operator+=(bigint &b);
    bigint operator-=(bigint &b);
```

```
bigint bigint::operator+(bigint &b)
{

    return result;
}

bigint bigint::operator-(bigint &b)
{
    /*先进行诊断，判断是否需要转诊*/
    if (b.sign == N && sign == P)
    {
        bigint tmpB = b;
        tmpB.sign = P;
        return((*this)+ tmpB);
    }
    else if (sign == N && b.sign == P)
    {
        bigint tmpB = b;
        tmpB.sign = N;
        return((*this)+ tmpB);
    }
```

```
    bigint operator++();
    bigint operator++(int);
    bigint operator--();
    bigint operator--(int);
    bool operator>(const bigint &b);
    bool operator>=(const bigint &b);
    bool operator<(const bigint &b);
    bool operator<=(const bigint &b);
    bool operator==(const bigint &b);
    bool operator!=(const bigint &b);
    operator string();//转换成2939或-23455
    friend ostream& operator<<(ostream& out,
bigint& b);
    friend istream& operator >> (istream& in,
bigint& a);
};
istream& operator >> (istream& in, bigint& a);
ostream& operator<<(ostream& out, bigint& b);

}
    else
    {
        lenMax = lenMin = len;
        if (content > b.content)
        {
            p1 = &(content[len - 1]);
            p2 = &((b.content)[b.len - 1]);
        }
        else
        {
            p1 = &((b.content)[b.len - 1]);
            p2 = &(content[len - 1]);
        }
    }


    /*判断结果的正负，还留在-函数中的只有双正或双
负两种情况，分别存在a>b以及b>a两种，
    由于result初始化sign为P，因此只需要设置sign为
N的情况*/

    if (sign == N&&b.sign == N)
    {
        if (p1 != &((b.content)[b.len-1]))//a更
大
            result.sign = N;
    }
    else
        if (p1 == &((b.content)[b.len-1]))//b更
大
            result.sign = N;

    result.len = lenMax;//两数相减的位数最多为两
者较长的数的位数
    result.content.resize(result.len +
1);//content大小为len+1
    (result.content)[result.len] = '\0';
```

```
    /*两个负整数或两个正整数的减处理*/
    bigint result;
    char *p1, *p2;/*p1指向被减数(较大)，p2指向
减数（较小）   备注：被减数-减数=差*/
    unsigned int lenMin = 0, lenMax = 0;
    if (len < b.len)
    {
        lenMin = len;
        lenMax = b.len;
        p1 = &((b.content)[b.len - 1]);
        p2 = &(content[len - 1]);
    }
    else if (len >b.len)
    {
        lenMin = b.len;
        lenMax = len;
        p1 = &(content[len - 1]);
        p2 = &((b.content)[b.len - 1]);
        if (tmpResult < 0)
        {
            tmpResult += 10;
            borrow = 1;
        }
        else
            borrow = 0;
        *pr = tmpResult + '0';
        pr--;
        i++;
    }
    /*两数相减会存在结果前面都为0的情况，要把
这串0去掉*/
    pr++;
    while (*pr == '0'&&zeroNum<lenMax)
    {
        zeroNum++;
        pr++;
    }
    result.content.assign(result.content,
zeroNum, lenMax-zeroNum);//把第一位去掉
    result.len-=zeroNum;
    return result;
}

bigint bigint::operator*(bigint &b)
{
    bigint result;

    /*确定符号位*/
    if (sign != b.sign)
        result.sign = N;

    /*确定长度*/
    result.len = len + b.len;//两者相乘位数最
多为两者位数之和
    result.content.resize(result.len + 1);//
多一位存储尾零
```

```
    /*用指针比用数组形式找效率更高*/
    char *pr = &((result.content)[result.len - 1]);
    unsigned int i = 0, borrow = 0,zeroNum=0;
    int tmpResult = 0;//必须用有符号型int! !
    while (i < lenMax)
    {
        if (i < lenMin)
        {
            tmpResult = *p1-*p2 - borrow;
            p1--;
            p2--;
        }
        else
        {
            tmpResult = *p1-'0' - borrow;//小心'0'
            p1--;
        }
    + carry;

            (*pr) += (tmpResult % 10);
            pTmp2 = pr;
            while ((*pr) > 9 + '0')
            {
                *pr = (*pr-'0') % 10+'0';
                pr--;
                (*pr)++;
            }
            pr = pTmp2;
            carry = tmpResult / 10;
            pb--;
            pr--;
        }
        (*pr) += carry;
        while ((*pr) > 9 + '0')
        {
            *pr = (*pr - '0') % 10 + '0';
            pr--;
            (*pr)++;
        }
        pTmp--;
        pa--;
    }
    if (result.content[0] == '0')
        result.content.assign(result.content, 1, result.len - 1);
    return result;
}

bigint bigint::operator/(bigint &b)
{
    bigint result;
    /*确定符号位*/
    if (sign != b.sign)
        result.sign = N;
    result.content.assign(result.len,'0');//存放数据的区域用'0'填充
    result.content[result.len] = '\0';


    char* pa = &(content[len - 1]);
    char* pb = &((b.content)[b.len - 1]);
    char* pr = &((result.content)[result.len - 1]);
    char* pTmp = pr,*pTmp2=pr,*pbTmp=pb;
    int carry = 0;//进位
    unsigned int tmpResult = 0;
    for (unsigned int i = 0; i < len; i++)
    {
        pr = pTmp;
        pb = pbTmp;
        for (unsigned int j = 0; j < b.len; j++)
        {
            tmpResult = (*pa - '0')*(*pb - '0') bigint tmpResult;//中间值

    if (len == b.len)
    {
        int count = 0;
        while (tmpA.len >b.len || tmpA.len == b.len&&tmpA.content > b.content)
        {
            if (b.sign == P)
                tmpA = tmpA - b;//不能单纯的-，因为正负是不确定的，需要分类讨论
            else
                tmpA = tmpA + b;
            tmpA.len = tmpA.content.length();
            count++;
        }

        result.content[0] = count+'0';
        return result;
    }

    bigint tmpB = b;
    tmpB.sign = P;
    int numZeroAdd = len - b.len - 1;//补充的0的个数
    while (tmpA.len>b.len||tmpA.len==b.len&&tmpA.content >b.content)
    {
        tmpResult.content.resize(numZeroAdd + 3);
        tmpResult.len = numZeroAdd + 2;

    tmpResult.content.assign(tmpResult.len, '0');
        tmpB = b;
        tmpB.content.append(numZeroAdd--, '0');
```

装 订 线

```
    /*特殊情况处理，简化算法*/
    if (len < b.len || len == b.len &&
content<=b.content)
    {
        result.len = 1;
        result.content = "0";
        if (content == b.content && len==b.len)
            result.content = "1";
        return result;
    }

    result.len = len - b.len + 1;//确定商的最大长
度
    result.content.assign(result.len, '0');//将
result赋为0串
    bigint tmpA = *this;//在减的过程中会改变，因
此需要一个替身
    tmpA.sign = P;//运算过程中是把正负隔离的，所
以用正整数来算
    {
    bigint result;

    /*确定符号位*/
    if (sign != b.sign)
        result.sign = N;

    int tmpSign = sign, tmpSignB = b.sign;//将
*this，b的符号位先存下来
    sign = P;
    b.sign = P;
    /*特殊情况处理，简化算法*/
    if (b>(*this))
    {
        result.len = len;
        result.content = content;
        result.sign = tmpSign;
        return result;
    }

    result.len = len - b.len + 1;//确定商的最大长
度
    result.content.assign(result.len, '0');//将
result赋为0串
    bigint tmpA = *this;//在减的过程中会改变，因
此需要一个替身

    if (len == b.len)
    {
        while (tmpA>=b)
        {
            tmpA = tmpA - b;
            tmpA.len = tmpA.content.length();
        }
        result.content = tmpA.content;
        result.len=tmpA.len;
        b.sign = tmpSignB;
        sign = tmpSign;//符号位复原
        tmpB.len = tmpB.content.length();
        if (tmpB > tmpA)//两者都是正的，可以
直接比较
            continue;
        int count = 0;
        while (tmpA>=tmpB)
        {
            tmpA = tmpA - tmpB;
            tmpA.len = tmpA.content.length();
            count++;
        }
        tmpResult.content[1] = count % 10+'0';
        tmpResult.content[0] = count / 10+'0';
        result =result+ tmpResult;
    }
    return result;
}

bigint bigint::operator%(bigint &b)
}
    result.len = tmpA.len;
    result.content = tmpA.content;
    b.sign = tmpSignB;
    sign = tmpSign;
    return result;
}
bigint bigint::operator++()//++a
{
    char *p = &(content[len - 1]);//指向最后
一位
    (*p)++;
    while ((*p) > 9 + '0')
    {
        (*p) = (*p) % 10 + '0';
        p--;
        if (p < &(content[0]))
        {
            content.replace(1, len - 1,
content);
            p++;
            (*p) = '0';
        }
        (*p)++;
    }
    bigint result = *this;
    return result;
}
bigint bigint::operator--()//--a
{
    char *p = &(content[len - 1]);//指向最后
一位
    (*p)--;
    while ((*p)< '0')
    {
        (*p) = (*p)+10;
```

装 订 线

```
        return result;
    }
    bigint tmpB = b;
    int numZeroAdd = len - b.len - 1;//补充的0的个数
    int i = 0;
    while (tmpA>=b)
    {
        tmpB = b;
        tmpB.content.append(numZeroAdd--, '0');
        tmpB.len = tmpB.content.length();
        if (tmpB > tmpA)//两者都是正的，可以直接比较
            continue;
        while (tmpA >= tmpB)
        {
            tmpA = tmpA - tmpB;
            tmpA.len = tmpA.content.length();
        }
#include<iostream>
#include<Windows.h>
#include<iomanip>
#include"90-b5-adv.h"
using namespace std;
int main()
{
    cout << "大数运算测试用例数据以及结果：" << endl;
    cout << "-------------------------------------------------------------------" << endl;
    cout << "测试大数构造：" << endl << endl;
    bigint a, b = "123465789898968988388383838";
    cout << "【无参构造a】,\n规则：无参构造初始化为0，因此输出a应为0，实际输出为：a=" << a << endl;;
    cout << "【有参数构造b】,\n应该输出b=\"123465789898968988388383838\",\n实际输出为:b=" << b << endl;

    cout << "【加减数据测试项：】" << endl;
    bigint c, d;
    cin >> c >> d;
    cout << endl << endl;

    cout << "【测试大数相加】：" << endl;
    cout << "c+d=" << c + d << endl;

    cout << "【测试大数相减】：" << endl;
    cout << "c-d=" << c - d << endl;
    cout << "d-c=" << d - c << endl;

    cout << "【测试正号运算符】：" << endl;
    cout << "+c=" << +c << endl;
    cout << "【测试负号运算符】：" << endl;
    cout << "-d=" << -d << endl;

    cout << "【测试赋值运算符】：" << endl;
        p--;
        (*p)--;
    }
    if (content[0] == '0')
    {
        content.assign(content, 0, len - 1);
        len--;
    }
    bigint result = *this;
    return result;
}
```

```
d) << endl;
    cout << "如果c==d,则返回1,实际返回：" << (c == d) << endl;
    cout << "如果c!=d,则返回1,实际返回：" << (c != d) << endl;

    cout << endl << endl;

    cout << "【乘除模以及赋值数据测试项：】" << endl;
    bigint g, h;
    cin >> g >> h;
    LARGE_INTEGER tick, fc_begin, fc_end;
    QueryPerformanceFrequency(&tick);
    QueryPerformanceCounter(&fc_begin);

    cout << "【测试大数相乘】：" << endl;
    cout << "g*h=" << g*h << endl;

    QueryPerformanceCounter(&fc_end);
    cout << "时钟频率：" << double(tick.QuadPart) / 1024 / 1024 << "GHz" << endl;
    cout << "时钟计数：" << double(fc_end.QuadPart - fc_begin.QuadPart) << endl;
    cout << setprecision(6) << double(fc_end.QuadPart - fc_begin.QuadPart) / double(tick.QuadPart) << "秒" << endl;

    cout << "【测试大数相除】：" << endl;
    cout << "g/h=" << g / h << endl;

    QueryPerformanceCounter(&fc_end);
    cout << "时钟频率：" << double(tick.QuadPart) / 1024 / 1024 << "GHz" << endl;
    cout << "时钟计数：" << 
```

装 订 线

```
    bigint e, f;
    e = c;
    cout << "e=c,则e=" << e << endl;
    cout << "f=(e=d),则f=" << f << endl;

    cout << "【测试++/--运算符】：" << endl;
    cout << "c++ =" << (c++) << endl;
    cout << "c-- =" << (c--) << endl;
    cout << "++c =" << (++c) << endl;
    cout << "--c =" << (--c) << endl;

    cout << "【测试比较运算符:】 " << endl;
    cout << "如果c>d, 则返回1,实际返回：" << (c > d) << endl;
    cout << "如果c>=d,则返回1,实际返回：" << (c >= d) << endl;
    cout << "如果c<d, 则返回1,实际返回：" << (c < d) << endl;
    cout << "如果c<=d,则返回1,实际返回：" << (c <=
```

```
double(fc_end.QuadPart - fc_begin.QuadPart) << endl;
    cout << setprecision(6) << double(fc_end.QuadPart - fc_begin.QuadPart) / double(tick.QuadPart) << "秒" << endl;

    cout << "【测试大数取模】：" << endl;
    cout << "g%h=" << g%h << endl;
    cout << "h%g=" << h%g << endl;

    cout << "【测试完成】" << endl;
    return 0;
}
```

装 订 线