长整型运算实验报告

题目：编制一个进行长整型运算的程序

班级：F1702128　姓名：沈嘉欢　学号：517021910872　完成日期：2018 年 9 月 21 日

一、 需求分析

本程序可实现任意长度整数的加、减、乘、除、乘方、阶乘运算。

用户可以通过 `iostream` 流进行整数的输入输出，通过类似于内置整数类的方式进行数值运算。

输入测试数据`-987 666`，可以得到输出的结果（由于输出结果过于庞大，此处仅展示部分）：

```
-321
-1653
-65,7342
-1
164,1434,9562,...,3641,2409
10,1063,2056,...,0000,0000
```

二、 概要设计

对于高精度整数运算问题，可以采用双向链表的方式对数值进行存储：

- 对每个整数创建一个 `BigInt` 对象，包含指向 `LinkedList` 私有结构体的首尾指针及整数长度；
- 头结点数据域记录数值的正负，`0` 表示非负整数，`1` 表示负整数；
- 结点由低位向高位存储，即头结点的 `next` 指针指向整数的个位，以此类推。

2.1 比较

对于两个 `BigInt` 对象进行比较，只需实现大于和等于的判定。

1. （大于的判定）若 $a > b$，当且仅当满足以下任一条件：

   (1) $a >= 0, b < 0$；

   (2) $a >= 0, b >= 0$，a 比 b 长；

   (3) $a >= 0, b >= 0$，a 与 b 长度相等，从最高位逐一比较，前 k 位相等，第 $k+1$ 位 a 的值比 b 大；

   (4) $a >= 0, b >= 0$，a 比 b 短；

(5) $a >= 0, b >= 0$，a 与 b 长度相等，从最高位逐一比较，前 k 位相等，第 $k+1$ 位 a 的值比 b 小；

2.（等于的判定）若 $a = b$，当且仅当满足 a 与 b 长度相等，每一位的值也相等。

有了“>”、“=”的比较方式，可以容易地得出“<”、“≤”、“≥”、“≠”的比较方式，此处不再说明。

2.2 加法

对于两个 `BigInt` 对象执行加法，有以下步骤：

1. 预处理：
   (1) 若 a 与 b 同号，计算 $|a| + |b|$ 再补充符号；
   (2) 若 $a < 0, b \geq 0$，计算 $b - (-a)$；
   (3) 若 $b < 0, a \geq 0$，计算 $a - (-b)$；
2. 将其链表的每个元素对应相加；
3. 从低到高处理进位。

2.3 减法

对于两个 `BigInt` 对象执行减法，有以下步骤：

1. 预处理：
   (1) 若 $a < 0$，计算 $-((-a) + b)$；
   (2) 若 $b < 0$，计算 $a + (-b)$；
   (3) 若 $a \geq 0, b \geq 0, a < b$，计算 $-(b - a)$；
2. 将其链表的每个元素对应相减；
3. 从低到高处理退位。

2.4 乘法

对于两个 `BigInt` 对象执行乘法，有以下步骤：

1. 符号判断：若 a 与 b 同号，结果为非负，反之结果为负；
2. 将 a 的第 i 位和 b 的第 j 位相乘的值加到结果的第 $i + j$ 位；
3. 从低到高处理进位。

2.5 除法

对于两个 BigInt 对象执行除法，有以下步骤：

1. 符号判断：若 a 与 b 同号，结果为非负，反之结果为负；
2. 记 $c = |a|, d = |b|$，e 为计算结果，计算 a/b 时，反复寻找满足 $c \geq d \cdot 10^k$ 的最大的 k，计算 $c = c - d \cdot 10^k$ 直至 $c - d \cdot 10^k < 0$，每次使 $e = e + 10^k$；
3. 对 e 添加符号。

有如下约定：

- 除数不能为 0；
- 出现不能整除的情况时，向 0 取整。

2.6 乘方、阶乘

对于两个 BigInt 对象执行乘方或阶乘，只需反复利用乘法运算即可。

2.7 大整数类 BigInt 基本操作

```
BigInt()
```

操作结果：生成一个空的大整数。

```
BigInt(num)
```

初始条件：num 为现有 string 对象或 BigInt 对象或内置整数类型。
操作结果：生成一个值为 num 的大整数。

```
~BigInt()
```

初始条件：BigInt 对象已建立。
操作结果：销毁 BigInt 对象。

```
operator+(first, second)
```

初始条件：first、second 两个 BigInt 对象已建立。
操作结果：获得两个对象之和。

```
operator-(first, second)
```

初始条件：first、second 两个 BigInt 对象已建立。
操作结果：获得两个对象之差。

```
operator*(first, second)
```

初始条件：first、second 两个 BigInt 对象已建立。
操作结果：获得两个对象之积。

```
operator/(first, second)
```

初始条件：first、second 两个 BigInt 对象已建立。
操作结果：获得两个对象之商。

```
power(first, second)
```

初始条件：first、second 两个 BigInt 对象已建立。
操作结果：获得对象的乘方。

```
factorial(num)
```

初始条件：num 对象已建立。
操作结果：获得对象的阶乘。

2.8 模块说明

本程序分为两个模块：

- 主程序模块：测试数据的输入输出；
- BigInt 模块，定义、实现大整数类。

三、 详细设计

3.1 BigInt 类声明

由于代码逻辑比较清晰，因此没有过多的代码说明。

```
//
// BigInt.h
// BigInt
//
// Created by 沈嘉欢 on 2018/9/21.
// Copyright © 2018 沈嘉欢. All rights reserved.
//

#ifndef BigInt_h
#define BigInt_h

#include <iostream>

class BigInt {
```

```
private:
    // 内嵌双向链表
    struct LinkedList {
        int data;
        LinkedList *front;
        LinkedList *next;
        LinkedList(int num=0, LinkedList *front=nullptr): data(num), front(front), next(nullptr)
            {};
        ~LinkedList() {
            delete next;
        }
    };
    LinkedList *header, *rear;
    int length;
    static BigInt abs(const BigInt &);
    // 处理进位、退位、去除最高位的0
    void simplify();
public:
    BigInt();
    BigInt(const std::string &);
    BigInt(const BigInt &);
    BigInt(long long);
    ~BigInt() {
        delete header;
    }
    friend bool operator>(const BigInt &, const BigInt &);
    friend bool operator<(const BigInt &, const BigInt &);
    friend bool operator>=(const BigInt &, const BigInt &);
    friend bool operator<=(const BigInt &, const BigInt &);
    friend bool operator==(const BigInt &, const BigInt &);
    friend bool operator!=(const BigInt &, const BigInt &);
    friend BigInt operator+(const BigInt &, const BigInt &);
    friend BigInt operator-(const BigInt &, const BigInt &);
    friend BigInt operator-(const BigInt &);
    friend BigInt operator*(const BigInt &, const BigInt &);
    friend BigInt operator/(const BigInt &, const BigInt &);
    BigInt &operator=(const std::string &);
    BigInt &operator=(const BigInt &);
    BigInt &operator=(long long);
    BigInt &operator+=(const BigInt &);
    BigInt &operator-=(const BigInt &);
    BigInt &operator*=(const BigInt &);
    BigInt &operator/=(const BigInt &);
    BigInt &operator++();
    BigInt operator++(int);
    BigInt &operator--();
    BigInt operator--(int);
```

```
    // 抽象出拷贝构造函数和赋值运算的共同代码
    friend void copy(BigInt &, const std::string &);
    friend void copy(BigInt &, const BigInt &);
    friend void copy(BigInt &, long long);
    friend BigInt power(const BigInt &, const BigInt &);
    friend BigInt factorial(const BigInt &);
    friend std::istream &operator>>(std::istream &, BigInt &);
    friend std::ostream &operator<<(std::ostream &, const BigInt &);
};

#endif /* BigInt_h */
```

3.2 BigInt 类实现

由于代码逻辑比较清晰，因此没有过多的代码说明。

```
//
// BigInt.cpp
// BigInt
//
// Created by 沈嘉欢 on 2018/9/21.
// Copyright © 2018 沈嘉欢. All rights reserved.
//

#include <iostream>
#include <string>
#include <climits>
#include "BigInt.h"

BigInt::BigInt() {
    header = new LinkedList();
    rear = header;
    length = 0;
}

BigInt::BigInt(const std::string &num) {
    assert(num.length() != 0);
    copy(*this, num);
}

BigInt::BigInt(const BigInt &num) {
    copy(*this, num);
}

BigInt::BigInt(long long num) {
    // 考虑到LLONG_MIN取绝对值会溢出，禁止这样的赋值
```

```
    assert(num != LLONG_MIN);
    copy(*this, num);
}

BigInt &BigInt::operator=(const std::string &num) {
    assert(num.length() != 0);
    delete this->header;
    copy(*this, num);
    return *this;
}

BigInt &BigInt::operator=(const BigInt &num) {
    if (this == &num) {
        return *this;
    }
    delete this->header;
    copy(*this, num);
    return *this;
}

BigInt &BigInt::operator=(long long num) {
    // 考虑到LLONG_MIN取绝对值会溢出，禁止这样的赋值
    assert(num != LLONG_MIN);
    delete this->header;
    copy(*this, num);
    return *this;
}

bool operator>(const BigInt &first, const BigInt &second) {
    if (first.header->data == 0 && second.header->data == 1) {
        return true;
    } else if (first.header->data == 1 && second.header->data == 0) {
        return false;
    }
    if (first.length != second.length) {
        if (first.header->data == 0) {
            return first.length > second.length;
        } else {
            return first.length < second.length;
        }
    }
    BigInt::LinkedList *p1 = first.rear, *p2 = second.rear;
    while (p1 != first.header && p2 != second.header) {
        if (p1->data != p2->data) {
            if (first.header->data == 0) {
                return p1->data > p2->data;
            } else {
```

```
            return p1->data < p2->data;
        }
    }
    p1 = p1->front;
    p2 = p2->front;
  }
  return false;
}

bool operator<(const BigInt &first, const BigInt &second){
  return !(first >= second);
}



bool operator==(const BigInt &first, const BigInt &second) {
  if (first.length != second.length) {
    return false;
  }
  if (first.header->data != second.header->data) {
    return false;
  }
  BigInt::LinkedList *p1 = first.rear, *p2 = second.rear;
  while (p1 != first.header && p2 != second.header) {
    if (p1->data != p2->data) {
      return false;
    }
    p1 = p1->front;
    p2 = p2->front;
  }
  return true;
}

bool operator!=(const BigInt &first, const BigInt &second) {
  return !(first == second);
}

bool operator>=(const BigInt &first, const BigInt &second) {
  return first > second || first == second;
}

bool operator<=(const BigInt &first, const BigInt &second) {
  return first < second || first == second;
}

BigInt BigInt::abs(const BigInt &num) {
  BigInt ans = num;
  ans.header->data = 0;
```

```
    return ans;
}

BigInt operator+(const BigInt &first, const BigInt &second) {
    BigInt ans;
    if (first.header->data == second.header->data) {
        ans.header->data = first.header->data;
    } else {
        if (first.header->data == 1) {
            return second - -first;
        } else {
            return first - -second;
        }
    }
    BigInt::LinkedList *p = first.header, *q = second.header, *r = ans.header;
    while (p->next && q->next) {
        r->next = new BigInt::LinkedList(p->next->data + q->next->data, r);
        ans.rear = r->next;
        p = p->next;
        q = q->next;
        r = r->next;
        ++ans.length;
    }
    while (p->next) {
        r->next = new BigInt::LinkedList(p->next->data, r);
        ans.rear = r->next;
        p = p->next;
        r = r->next;
        ++ans.length;
    }
    while (q->next) {
        r->next = new BigInt::LinkedList(q->next->data, r);
        ans.rear = r->next;
        q = q->next;
        r = r->next;
        ++ans.length;
    }
    ans.simplify();
    return ans;
}

BigInt operator-(const BigInt &num) {
    BigInt ans = num;
    if (!(ans.length == 1 && ans.header->next->data == 0)) {
        ans.header->data = 1 - ans.header->data;
    }
    return ans;
```

```
}

BigInt operator-(const BigInt &first, const BigInt &second) {
    if (first.header->data == 1) {
        return -(-first + second);
    }
    if (second.header->data == 1) {
        return first + (-second);
    }
    if (first < second) {
        return -(second - first);
    }
    BigInt ans;
    ans.header->data = 0;
    BigInt::LinkedList *p = first.header, *q = second.header, *r = ans.header;
    while (p->next && q->next) {
        r->next = new BigInt::LinkedList(p->next->data - q->next->data, r);
        ans.rear = r->next;
        p = p->next;
        q = q->next;
        r = r->next;
        ++ans.length;
    }
    while (p->next) {
        r->next = new BigInt::LinkedList(p->next->data, r);
        ans.rear = r->next;
        p = p->next;
        r = r->next;
        ++ans.length;
    }
    ans.simplify();
    return ans;
}

BigInt operator*(const BigInt &first, const BigInt &second) {
    BigInt ans;
    ans.header->data = (first.header->data == second.header->data) ? 0 : 1;
    ans.length = first.length + second.length;
    BigInt::LinkedList *p = first.header->next, *q = second.header->next, *r = ans.header, *s =
        ans.header;
    for (int i = 1; i <= ans.length; ++i) {
        r->next = new BigInt::LinkedList(0, r);
        r = r->next;
    }
    ans.rear = r;
    r = ans.header->next;
    while (p) {
```

```
        s = r;
        q = second.header->next;
        while (q) {
            s->data += p->data * q->data;
            s = s->next;
            q = q->next;
        }
        r = r->next;
        p = p->next;
    }
    ans.simplify();
    return ans;
}

BigInt operator/(const BigInt &first, const BigInt &second) {
    assert(second != 0);
    BigInt ans;
    ans.length = first.length;
    BigInt::LinkedList *p = ans.header;
    for (int i = 1; i <= ans.length; ++i) {
        p->next = new BigInt::LinkedList(0, p);
        p = p->next;
    }
    ans.rear = p;
    BigInt remain = BigInt::abs(first);
    BigInt subtrahend = BigInt::abs(second);
    while (true) {
        BigInt stdSubtrahend = 1;
        while (remain >= stdSubtrahend * 10 * subtrahend) {
            stdSubtrahend *= 10;
        }
        while (remain >= stdSubtrahend * subtrahend) {
            ans = ans + stdSubtrahend;
            remain = remain - stdSubtrahend * subtrahend;
        }
        if (stdSubtrahend == 1) {
            break;
        }
    }
    ans.header->data = (first.header->data == second.header->data) ? 0 : 1;
    ans.simplify();
    if (ans.length == 1 && ans.header->next->data == 0) {
        ans.header->data = 0;
    }
    return ans;
}
```

```
BigInt &BigInt::operator+=(const BigInt &second) {
    BigInt tmp = *this + second;
    *this = tmp;
    return *this;
}

BigInt &BigInt::operator-=(const BigInt &second) {
    BigInt tmp = *this - second;
    *this = tmp;
    return *this;
}

BigInt &BigInt::operator*=(const BigInt &second) {
    BigInt tmp = *this * second;
    *this = tmp;
    return *this;
}

BigInt &BigInt::operator/=(const BigInt &second) {
    BigInt tmp = *this / second;
    *this = tmp;
    return *this;
}

BigInt &BigInt::operator++() {
    *this += 1;
    return *this;
}

BigInt BigInt::operator++(int x) {
    BigInt tmp = *this;
    ++(*this);
    return tmp;
}

BigInt &BigInt::operator--() {
    *this -= 1;
    return *this;
}

BigInt BigInt::operator--(int x) {
    BigInt tmp = *this;
    --(*this);
    return tmp;
}

void copy(BigInt &first, const std::string &second) {
```

```
    first.header = new BigInt::LinkedList();
    first.rear = first.header;
    first.length = 0;
    auto begin = second.rbegin();
    auto end = second.rend();
    if (*second.begin() == '-') {
        first.header->data = 1;
        assert(second.length() >= 1);
        end--;
    }
    BigInt::LinkedList *p = first.header;
    for (auto it = begin; it != end; ++it) {
        if (*it == ',') {
            continue;
        }
        p->next = new BigInt::LinkedList(*it - '0', p);
        first.rear = p->next;
        ++first.length;
        p = p->next;
    }
}

void copy(BigInt &first, const BigInt &second) {
    first.header = new BigInt::LinkedList();
    first.rear = first.header;
    first.length = second.length;
    first.header->data = second.header->data;
    BigInt::LinkedList *p = first.header, *q = second.header;
    while (q->next) {
        p->next = new BigInt::LinkedList(q->next->data, p);
        p = first.rear = p->next;
        q = q->next;
    }
}

void copy(BigInt &first, long long second) {
    first.header = new BigInt::LinkedList();
    first.rear = first.header;
    first.length = 0;
    if (second == 0) {
        first.rear = first.header->next = new BigInt::LinkedList(0);
        first.length = 1;
    } else {
        if (second < 0) {
            first.header->data = 1;
            second = -second;
        }
```

```
        BigInt::LinkedList *p = first.header;
        while (second) {
            p->next = new BigInt::LinkedList(second % 10, p);
            second /= 10;
            first.length++;
            p = first.rear = p->next;
        }
    }
}

BigInt power(const BigInt &first, const BigInt &second) {
    assert(second >= 0);
    BigInt ans = 1;
    for (BigInt i = 1; i <= second; ++i) {
        ans *= first;
    }
    return ans;
}

BigInt factorial(const BigInt &num) {
    assert(num >= 0);
    BigInt ans = 1;
    for (BigInt i = 1; i <= num; ++i) {
        ans *= i;
    }
    return ans;
}

void BigInt::simplify() {
    LinkedList *p = header->next;
    while (p->next) {
        p->data += 10;
        --p->next->data;
        p->next->data += p->data / 10;
        p->data %= 10;
        p = p->next;
    }
    if (p->data >= 10) {
        p->next = new LinkedList(p->data / 10, p);
        rear = p->next;
        ++length;
        p->data %= 10;
    }
    while (rear->data == 0 && length != 1) {
        LinkedList *tmp = rear->front;
        --length;
        tmp->next = nullptr;
```

```
        rear->front = nullptr;
        delete rear;
        rear = tmp;
    }
    if (length == 1 && header->next->data == 0) {
        header->data = 0;
    }
}

std::istream &operator>>(std::istream &in, BigInt &num) {
    std::string tmp;
    in >> tmp;
    for (auto it : tmp) {
        assert((it >= '0' && it <= '9') || it == ',' || it == '-');
    }
    num = tmp;
    return in;
}

std::ostream &operator<<(std::ostream &out, const BigInt &num) {
    if (num.header->data == 1) {
        out << '-';
    }
    BigInt::LinkedList *p = num.rear;
    int cnt = 0;
    while (p != num.header) {
        out << p->data;
        ++cnt;
        p = p->front;
        if ((num.length - cnt) % 4 == 0 && p != num.header) {
            out << ",";
        }
    }
    return out;
}
```

3.3 主程序模块

```
//
// main.cpp
// BigInt
//
// Created by 沈嘉欢 on 2018/9/21.
// Copyright © 2018 沈嘉欢. All rights reserved.
//
```

```cpp
#include <iostream>
#include "BigInt.h"

int main() {
    using std::cin;
    using std::cout;
    using std::endl;
    BigInt a, b;
    cin >> a >> b;
    cout << a + b << endl;
    cout << a - b << endl;
    cout << a * b << endl;
    cout << a / b << endl;
    cout << power(a, b) << endl;
    cout << factorial(b) << endl;
    return 0;
}
```

四、 调试分析

本程序基本实现了大部分的整数数值运算操作。利用运算符重载功能，在现有的代码基础上，可以容易地将代码扩展到其他运算。

五、 用户手册

本程序在 Apple LLVM version 10.0.0 (clang-1000.11.45.2) 环境编译通过，文件组织结构为：

```
BigInt
    main.cpp
    BigInt.h
    BitInt.cpp
```

使用编译命令 `$ clang++ -o BigInt main.cpp BigInt.cpp --std=c++11`，会生成名为 `BigInt` 的 Unix 可执行文件。

运行后，终端等待用户输入两个整数，并返回其加、减、乘、除、乘方及第一个数的阶乘。

六、 测试结果

```
$ clang++ -o BigInt main.cpp BigInt.cpp --std=c++11
$ ./BigInt
-987 666
-321
-1653
-65,7342
-1
164,1434,9562,...,3641,2409
10,1063,2056,...,0000,0000
$
```