计算机科学技术学院实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **课程名称** | 面向对象程序设计 | | | **学 号** |  |
| **实验项目** | 实验三 | | | **姓 名** |  |
| **学 时** | 2 | **项目性质** | 综合型 | **班 级** | 2405111B |
| **指导教师** | 蒋振刚 | **实验地点** | 实训楼424 | **日 期** | 2024年4月14日 |
| 1. **实验目的和要求**   第4个实验：大整数类。  目的：练习运算符重载的实现和具体应用，进一步练习类的设计和使用，练习对数据的操控能力。  要求：实现不超过100位的大整数类型，支持  1）流输入，输出；  2）赋值；  3）加、减，乘，除运算；  4）与int，float，double的混合运算。 | | | | | |
| 1. **实验环境**   系统：fedora 42  环境：g++ | | | | | |
| 1. **实验内容与过程**   **BigInteger.h:**  **#ifndef BIGINTEGER\_H**  **#define BIGINTEGER\_H**  **#include <iostream>**  **#include <string>**  **#include <vector>**  **#include <algorithm>**  **#include <cstring>**  **class BigInteger {**  **public:**  **BigInteger();**  **BigInteger(const char\* str);**  **BigInteger(const BigInteger& other);**  **~BigInteger();**  **BigInteger& operator=(const BigInteger& other);**  **BigInteger operator+(const BigInteger& other) const;**  **BigInteger operator-(const BigInteger& other) const;**  **BigInteger operator\*(const BigInteger& other) const;**  **BigInteger operator/(const BigInteger& other) const;**  **bool operator==(const BigInteger& other) const;**  **bool operator!=(const BigInteger& other) const;**  **bool operator<(const BigInteger& other) const;**  **bool operator>(const BigInteger& other) const;**  **friend std::ostream& operator<<(std::ostream& os, const BigInteger& bigInt);**  **friend std::istream& operator>>(std::istream& is, BigInteger& bigInt);**  **void setDecimalPlaces(size\_t places);**  **size\_t getDecimalPlaces() const;**  **void roundToPlaces(size\_t places);**  **private:**  **std::vector<int> digits; // 存储数字的反向表示**  **bool isNegative;           // 表示正负**  **size\_t decimalPlaces;      // 小数位数**  **void removeLeadingZeros(); // 移除前导零**  **void removeTrailingZeros(); // 移除尾部多余的零**  **};**  **BigInteger::BigInteger() : isNegative(false), decimalPlaces(0) {**  **digits.push\_back(0); // 默认值为0**  **}**  **BigInteger::BigInteger(const char\* str) : isNegative(false), decimalPlaces(0) {**  **if (str == nullptr || str[0] == '\0') {**  **digits.push\_back(0);**  **return;**  **}**  **int start = 0;**  **if (str[0] == '-') {**  **isNegative = true;**  **start = 1;**  **} else if (str[0] == '+') {**  **start = 1;**  **}**  **// 查找小数点位置**  **const char\* decimalPoint = strchr(str + start, '.');**  **size\_t decimalPos = 0;**    **if (decimalPoint != nullptr) {**  **decimalPos = decimalPoint - str;**  **decimalPlaces = strlen(decimalPoint + 1);**  **}**  **// 反向存储数字部分**  **for (int i = strlen(str) - 1; i >= start; --i) {**  **if (str[i] == '.') continue;**    **if (str[i] >= '0' && str[i] <= '9') {**  **digits.push\_back(str[i] - '0');**  **} else {**  **// 非法字符，设为0**  **digits.clear();**  **digits.push\_back(0);**  **isNegative = false;**  **decimalPlaces = 0;**  **return;**  **}**  **}**  **removeLeadingZeros();**  **removeTrailingZeros();**    **if (digits.size() == 1 && digits[0] == 0) {**  **isNegative = false;**  **decimalPlaces = 0;**  **}**  **}**  **BigInteger::BigInteger(const BigInteger& other) : digits(other.digits), isNegative(other.isNegative), decimalPlaces(other.decimalPlaces) {**  **}**  **BigInteger::~BigInteger() {**  **// 向量会自动清理，不需要特殊处理**  **}**  **void BigInteger::removeLeadingZeros() {**  **while (digits.size() > 1 && digits.back() == 0) {**  **digits.pop\_back();**  **}**  **}**  **void BigInteger::removeTrailingZeros() {**  **while (decimalPlaces > 0 && !digits.empty() && digits[0] == 0) {**  **digits.erase(digits.begin());**  **--decimalPlaces;**  **}**  **}**  **BigInteger& BigInteger::operator=(const BigInteger& other) {**  **if (this != &other) {**  **digits = other.digits;**  **isNegative = other.isNegative;**  **decimalPlaces = other.decimalPlaces;**  **}**  **return \*this;**  **}**  **BigInteger BigInteger::operator+(const BigInteger& other) const {**  **// 如果符号不同，转为减法**  **if (isNegative != other.isNegative) {**  **BigInteger temp = other;**  **temp.isNegative = !temp.isNegative;**  **return \*this - temp;**  **}**  **BigInteger result;**  **result.digits.clear();**  **result.isNegative = isNegative; // 保持符号一致**  **// 对齐小数点**  **size\_t maxDecimalPlaces = std::max(decimalPlaces, other.decimalPlaces);**  **BigInteger a = \*this;**  **BigInteger b = other;**    **// 补齐小数位**  **while (a.decimalPlaces < maxDecimalPlaces) {**  **a.digits.insert(a.digits.begin(), 0);**  **a.decimalPlaces++;**  **}**  **while (b.decimalPlaces < maxDecimalPlaces) {**  **b.digits.insert(b.digits.begin(), 0);**  **b.decimalPlaces++;**  **}**  **int carry = 0;**  **size\_t maxLen = std::max(a.digits.size(), b.digits.size());**    **// 确保两个数字长度相同**  **while (a.digits.size() < maxLen) a.digits.push\_back(0);**  **while (b.digits.size() < maxLen) b.digits.push\_back(0);**  **// 逐位相加**  **for (size\_t i = 0; i < maxLen || carry; ++i) {**  **int sum = carry;**  **if (i < a.digits.size()) sum += a.digits[i];**  **if (i < b.digits.size()) sum += b.digits[i];**    **result.digits.push\_back(sum % 10);**  **carry = sum / 10;**  **}**  **result.decimalPlaces = maxDecimalPlaces;**  **result.removeLeadingZeros();**  **result.removeTrailingZeros();**  **// 特殊情况：如果结果为0**  **if (result.digits.size() == 1 && result.digits[0] == 0) {**  **result.isNegative = false;**  **result.decimalPlaces = 0;**  **}**  **return result;**  **}**  **BigInteger BigInteger::operator-(const BigInteger& other) const {**  **// 如果符号不同，转为加法**  **if (isNegative != other.isNegative) {**  **BigInteger temp = other;**  **temp.isNegative = !temp.isNegative;**  **return \*this + temp;**  **}**  **// 对齐小数点**  **size\_t maxDecimalPlaces = std::max(decimalPlaces, other.decimalPlaces);**  **BigInteger a = \*this;**  **BigInteger b = other;**    **// 补齐小数位**  **while (a.decimalPlaces < maxDecimalPlaces) {**  **a.digits.insert(a.digits.begin(), 0);**  **a.decimalPlaces++;**  **}**  **while (b.decimalPlaces < maxDecimalPlaces) {**  **b.digits.insert(b.digits.begin(), 0);**  **b.decimalPlaces++;**  **}**  **// 比较绝对值大小**  **bool thisLarger = (a.digits.size() > b.digits.size());**  **if (a.digits.size() == b.digits.size()) {**  **for (int i = a.digits.size() - 1; i >= 0; --i) {**  **if (a.digits[i] != b.digits[i]) {**  **thisLarger = (a.digits[i] > b.digits[i]);**  **break;**  **}**  **}**  **}**  **BigInteger result;**  **result.digits.clear();**  **result.decimalPlaces = maxDecimalPlaces;**  **// 确保被减数大于减数**  **const std::vector<int>& larger = thisLarger ? a.digits : b.digits;**  **const std::vector<int>& smaller = thisLarger ? b.digits : a.digits;**    **// 确定结果的符号**  **result.isNegative = isNegative ? thisLarger : !thisLarger;**  **int borrow = 0;**    **// 确保两个数字长度相同**  **std::vector<int> largerDigits = larger;**  **std::vector<int> smallerDigits = smaller;**  **while (smallerDigits.size() < largerDigits.size()) {**  **smallerDigits.push\_back(0);**  **}**  **// 逐位相减**  **for (size\_t i = 0; i < largerDigits.size(); ++i) {**  **int diff = largerDigits[i] - borrow - smallerDigits[i];**  **if (diff < 0) {**  **diff += 10;**  **borrow = 1;**  **} else {**  **borrow = 0;**  **}**  **result.digits.push\_back(diff);**  **}**  **result.removeLeadingZeros();**  **result.removeTrailingZeros();**  **// 如果结果为0**  **if (result.digits.size() == 1 && result.digits[0] == 0) {**  **result.isNegative = false;**  **result.decimalPlaces = 0;**  **}**  **return result;**  **}**  **BigInteger BigInteger::operator\*(const BigInteger& other) const {**  **BigInteger result;**  **result.digits.assign(digits.size() + other.digits.size(), 0);**  **result.isNegative = isNegative != other.isNegative;**    **// 计算结果的小数位数应该是两个操作数小数位数之和**  **result.decimalPlaces = decimalPlaces + other.decimalPlaces;**  **// 执行乘法运算**  **for (size\_t i = 0; i < digits.size(); i++) {**  **int carry = 0;**  **for (size\_t j = 0; j < other.digits.size() || carry; j++) {**  **int current = result.digits[i + j] +**  **digits[i] \* (j < other.digits.size() ? other.digits[j] : 0) +**  **carry;**  **result.digits[i + j] = current % 10;**  **carry = current / 10;**  **}**  **}**  **result.removeLeadingZeros();**  **result.removeTrailingZeros();**    **// 如果结果是0，确保它是正数且没有小数位**  **if (result.digits.size() == 1 && result.digits[0] == 0) {**  **result.isNegative = false;**  **result.decimalPlaces = 0;**  **}**    **return result;**  **}**  **BigInteger BigInteger::operator/(const BigInteger& other) const {**  **// 除数为0，抛出异常**  **if (other.digits.size() == 1 && other.digits[0] == 0) {**  **throw std::runtime\_error("Division by zero");**  **}**  **// 被除数为0，结果为0**  **if (digits.size() == 1 && digits[0] == 0) {**  **return BigInteger();**  **}**  **// 将小数转换为整数进行运算**  **BigInteger a = \*this;**  **BigInteger b = other;**    **// 计算结果的小数位数**  **// 结果的小数位 = 被除数的小数位 - 除数的小数位 + 额外精度**  **size\_t extraPrecision = 4;  // 额外精度**  **size\_t resultDecimalPlaces = a.decimalPlaces + extraPrecision;**    **// 将除数转为整数**  **while (b.decimalPlaces > 0) {**  **a.digits.insert(a.digits.begin(), 0);  // 被除数相应左移**  **b.decimalPlaces--;**  **}**    **// 补充额外的精度位**  **for (size\_t i = 0; i < extraPrecision; i++) {**  **a.digits.insert(a.digits.begin(), 0);**  **}**  **BigInteger quotient;**  **quotient.digits.clear();**  **quotient.isNegative = isNegative != other.isNegative;**    **BigInteger remainder;**  **remainder.digits.clear();**    **// 执行除法运算**  **for (int i = a.digits.size() - 1; i >= 0; i--) {**  **remainder.digits.insert(remainder.digits.begin(), a.digits[i]);**  **remainder.removeLeadingZeros();**    **int q = 0;**  **BigInteger temp = b;**  **temp.isNegative = false;**    **while (!(remainder < temp) && remainder.digits.size() > 0) {**  **remainder = remainder - temp;**  **q++;**  **}**    **quotient.digits.push\_back(q);**  **}**    **// 反转结果**  **std::reverse(quotient.digits.begin(), quotient.digits.end());**  **quotient.removeLeadingZeros();**    **// 设置正确的小数位数**  **quotient.decimalPlaces = resultDecimalPlaces;**  **quotient.removeTrailingZeros();**    **return quotient;**  **}**  **bool BigInteger::operator==(const BigInteger& other) const {**  **if (isNegative != other.isNegative || digits.size() != other.digits.size() || decimalPlaces != other.decimalPlaces) {**  **return false;**  **}**    **for (size\_t i = 0; i < digits.size(); ++i) {**  **if (digits[i] != other.digits[i]) {**  **return false;**  **}**  **}**    **return true;**  **}**  **bool BigInteger::operator!=(const BigInteger& other) const {**  **return !(\*this == other);**  **}**  **bool BigInteger::operator<(const BigInteger& other) const {**  **// 如果符号不同，负数小于正数**  **if (isNegative != other.isNegative) {**  **return isNegative;**  **}**    **// 对齐小数点进行比较**  **BigInteger a = \*this;**  **BigInteger b = other;**    **// 补齐小数位**  **size\_t maxDecimalPlaces = std::max(a.decimalPlaces, b.decimalPlaces);**  **while (a.decimalPlaces < maxDecimalPlaces) {**  **a.digits.insert(a.digits.begin(), 0);**  **a.decimalPlaces++;**  **}**  **while (b.decimalPlaces < maxDecimalPlaces) {**  **b.digits.insert(b.digits.begin(), 0);**  **b.decimalPlaces++;**  **}**    **// 如果都是负数，则比较绝对值（绝对值大的实际上更小）**  **if (isNegative) {**  **if (a.digits.size() != b.digits.size()) {**  **return a.digits.size() > b.digits.size();**  **}**    **for (int i = a.digits.size() - 1; i >= 0; --i) {**  **if (a.digits[i] != b.digits[i]) {**  **return a.digits[i] > b.digits[i];**  **}**  **}**    **return false; // 相等的情况**  **}**    **// 如果都是正数，直接比较**  **if (a.digits.size() != b.digits.size()) {**  **return a.digits.size() < b.digits.size();**  **}**    **for (int i = a.digits.size() - 1; i >= 0; --i) {**  **if (a.digits[i] != b.digits[i]) {**  **return a.digits[i] < b.digits[i];**  **}**  **}**    **return false; // 相等的情况**  **}**  **bool BigInteger::operator>(const BigInteger& other) const {**  **return other < \*this;**  **}**  **void BigInteger::setDecimalPlaces(size\_t places) {**  **if (places > decimalPlaces) {**  **// 需要补零**  **size\_t zerosToAdd = places - decimalPlaces;**  **for (size\_t i = 0; i < zerosToAdd; ++i) {**  **digits.insert(digits.begin(), 0);**  **}**  **} else if (places < decimalPlaces) {**  **// 需要截断或四舍五入**  **roundToPlaces(places);**  **}**  **decimalPlaces = places;**  **}**  **size\_t BigInteger::getDecimalPlaces() const {**  **return decimalPlaces;**  **}**  **void BigInteger::roundToPlaces(size\_t places) {**  **if (places >= decimalPlaces) return;**    **// 判断需要四舍五入的位**  **if (places < decimalPlaces && places < digits.size()) {**  **if (digits[decimalPlaces - places - 1] >= 5) {**  **// 进位**  **int carry = 1;**  **size\_t pos = decimalPlaces - places;**  **while (pos < digits.size() && carry) {**  **digits[pos] += carry;**  **carry = digits[pos] / 10;**  **digits[pos] %= 10;**  **++pos;**  **}**  **if (carry) digits.push\_back(1);**  **}**  **}**  **// 移除多余的小数位**  **if (decimalPlaces > places) {**  **digits.erase(digits.begin(), digits.begin() + (decimalPlaces - places));**  **decimalPlaces = places;**  **}**  **}**  **std::ostream& operator<<(std::ostream& os, const BigInteger& bigInt) {**  **if (bigInt.digits.empty()) {**  **os << "0";**  **return os;**  **}**    **if (bigInt.isNegative) {**  **os << "-";**  **}**    **size\_t totalSize = bigInt.digits.size();**    **// 如果没有小数部分**  **if (bigInt.decimalPlaces == 0) {**  **for (int i = totalSize - 1; i >= 0; --i) {**  **os << bigInt.digits[i];**  **}**  **return os;**  **}**    **// 处理有小数部分的情况**  **size\_t integerPartSize = totalSize - bigInt.decimalPlaces;**    **// 输出整数部分**  **if (integerPartSize == 0) {**  **os << "0";**  **} else {**  **for (int i = totalSize - 1; i >= bigInt.decimalPlaces; --i) {**  **os << bigInt.digits[i];**  **}**  **}**    **// 输出小数部分**  **if (bigInt.decimalPlaces > 0) {**  **os << ".";**  **for (int i = bigInt.decimalPlaces - 1; i >= 0; --i) {**  **os << bigInt.digits[i];**  **}**  **}**    **return os;**  **}**  **std::istream& operator>>(std::istream& is, BigInteger& bigInt) {**  **std::string str;**  **is >> str;**  **bigInt = BigInteger(str.c\_str());**  **return is;**  **}**  **#endif**  **Main.cpp:**  **#include <iostream>**  **#include "include/BigInteger.h"**  **// 测试函数：展示两个 BigInteger 之间的各种操作**  **void testOperation(const char\* num1Str, const char\* num2Str) {**  **BigInteger num1(num1Str);**  **BigInteger num2(num2Str);**    **std::cout << "\n===== 测试: " << num1 << " 和 " << num2 << " =====\n";**    **// 测试加法**  **std::cout << num1 << " + " << num2 << " = " << (num1 + num2) << std::endl;**    **// 测试减法**  **std::cout << num1 << " - " << num2 << " = " << (num1 - num2) << std::endl;**  **std::cout << num2 << " - " << num1 << " = " << (num2 - num1) << std::endl;**    **// 测试乘法**  **std::cout << num1 << " \* " << num2 << " = " << (num1 \* num2) << std::endl;**    **// 测试除法（避免除以零）**  **if (!(num2 == BigInteger("0"))) {**  **std::cout << num1 << " / " << num2 << " = " << (num1 / num2) << std::endl;**  **} else {**  **std::cout << num1 << " / " << num2 << " = 错误: 除数不能为零" << std::endl;**  **}**    **// 测试比较运算符**  **std::cout << num1 << " == " << num2 << " ? " << (num1 == num2 ? "是" : "否") << std::endl;**  **std::cout << num1 << " != " << num2 << " ? " << (num1 != num2 ? "是" : "否") << std::endl;**  **std::cout << num1 << " < " << num2 << " ? " << (num1 < num2 ? "是" : "否") << std::endl;**  **std::cout << num1 << " > " << num2 << " ? " << (num1 > num2 ? "是" : "否") << std::endl;**  **}**  **// 测试输入输出流操作**  **void testIO() {**  **BigInteger num;**  **std::cout << "\n===== 测试输入/输出 =====\n";**  **std::cout << "请输入一个整数: ";**  **std::cin >> num;**  **std::cout << "您输入的数字是: " << num << std::endl;**  **}**  **// 测试一些大数计算**  **void testLargeNumbers() {**  **std::cout << "\n===== 测试大数计算 =====\n";**    **// 计算大数阶乘**  **BigInteger factorial("1");**  **int n = 20;  // 计算 20!**    **std::cout << n << "! = ";**  **for (int i = 2; i <= n; ++i) {**  **factorial = factorial \* BigInteger(std::to\_string(i).c\_str());**  **}**  **std::cout << factorial << std::endl;**    **// 大数加法**  **BigInteger large1("9999999999999999999999");**  **BigInteger large2("1");**  **std::cout << large1 << " + " << large2 << " = " << (large1 + large2) << std::endl;**  **}**  **// 添加测试小数运算的函数**  **void testDecimalOperations() {**  **std::cout << "\n===== 测试小数运算 =====\n";**    **// 测试基本小数**  **BigInteger num1("123.456");**  **BigInteger num2("78.90");**  **std::cout << "基本小数运算:\n";**  **std::cout << num1 << " + " << num2 << " = " << (num1 + num2) << std::endl;**  **std::cout << num1 << " - " << num2 << " = " << (num1 - num2) << std::endl;**  **std::cout << num1 << " \* " << num2 << " = " << (num1 \* num2) << std::endl;**  **std::cout << num1 << " / " << num2 << " = " << (num1 / num2) << std::endl;**    **// 测试不同精度小数**  **BigInteger num3("1.23456");**  **BigInteger num4("2.1");**  **std::cout << "\n不同精度小数运算:\n";**  **std::cout << num3 << " + " << num4 << " = " << (num3 + num4) << std::endl;**    **// 测试四舍五入**  **BigInteger num5("3.14159");**  **std::cout << "\n四舍五入测试:\n";**  **std::cout << "原数: " << num5 << std::endl;**  **num5.roundToPlaces(2);**  **std::cout << "保留2位小数: " << num5 << std::endl;**    **// 测试设置小数位数**  **BigInteger num6("1.5");**  **std::cout << "\n设置小数位数测试:\n";**  **std::cout << "原数: " << num6 << std::endl;**  **num6.setDecimalPlaces(3);**  **std::cout << "设置3位小数: " << num6 << std::endl;**    **// 测试负数小数**  **BigInteger num7("-12.34");**  **BigInteger num8("5.67");**  **std::cout << "\n负数小数运算:\n";**  **std::cout << num7 << " + " << num8 << " = " << (num7 + num8) << std::endl;**  **std::cout << num7 << " \* " << num8 << " = " << (num7 \* num8) << std::endl;**  **}**  **int main() {**    **// 测试基本操作**  **testOperation("123", "456");         // 正数与正数**  **testOperation("-123", "456");        // 负数与正数**  **testOperation("123", "-456");        // 正数与负数**  **testOperation("-123", "-456");       // 负数与负数**  **testOperation("0", "123");           // 零与非零**  **testOperation("1.23","4");         // 小数与整数**  **testOperation("123456789", "987654321");  // 大数测试**    **// 测试输入输出**  **testIO();**    **// 测试大数计算**  **testLargeNumbers();**    **// 小数测试**  **testDecimalOperations();**    **return 0;**  **}**  **图片包含 图示  AI 生成的内容可能不正确。** | | | | | |
| 1. **实验结果与分析**     **用字符串模拟高精度整数运算，核心思想包括：**  **用字符串表示大整数的每一位（从低位到高位倒着存），便于实现加减乘除；**  **用布尔值表示符号位（isNegative）；**  **运算符重载 + 辅助函数 实现类似内建整数的用法；**  **对于各类操作都支持“加减符号判断 + 数值处理”两部分逻辑分离；** | | | | | |
| 1. **实验心得**   在实验过程中，我通过使用字符串倒序存储数字位数的方式，从而实现了大整数的模拟计算。整个实验的实现过程中，我重点掌握了以下几个方面的知识和能力：  **1. 运算符重载与面向对象设计**  大整数类最基础的需求是：能够像普通数值一样使用加减乘除等操作。因此，我需要为 + - \* / % 运算符进行重载。通过学习与练习，我更加熟悉了 C++ 中类成员函数的定义、友元函数的使用、对象传参与返回等内容。此外，类中的构造函数、私有成员、辅助函数等，也让我对面向对象程序设计有了更深的理解。  **2. 字符串模拟高精度计算**  由于内置整型无法满足需求，我采用了字符串来保存大整数，并通过字符与整数之间的转换、按位相加/相减/乘除，模拟了小学数学中的“竖式计算”过程。这一过程虽然实现逻辑较复杂，但对我的算法设计与调试能力提出了很好的锻炼。  特别是在减法和除法的实现中，需要先判断符号，再处理绝对值大小关系，这让我理解了许多平时用不到但实现中必须考虑的边界问题。  **3. 运算符符号控制与负数处理**  在加减乘除等运算中，符号判断是实现的一个难点。通过布尔值标记负号并在计算前进行统一处理，我解决了符号错乱的问题。此外，为了让 BigInteger 类更加稳定，还实现了删除前导零、结果归一化、绝对值比较等辅助方法。 | | | | | |
| 1. **教师评语** | | | | | |
| 1. **实验成绩**   教师签名： 蒋振刚 批阅日期： 2024 年 4 月 14日 | | | | | |

注：项目性质为 演示型、验证型、设计型、综合型和创新型。