Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу дискретного анализа

Студент: А. В. Синявский
Преподаватель: Н. А. Зацепин
Группа: М8О-308Б-18
Дата:
Оценка:
Подпись:

Москва, 2020

Лабораторная работа № 6 по курсу дискретного анализа

Выполнил студент группы М80-308Б-18 МАИ *Синявский Андрей.*

Условие

Необходимо разработать программную библиотеку на языке C или C++, реализующую простейшие арифметические действия и проверку условий над целыми неотрицательными числами. На основании этой библиотеки нужно составить программу, выполняющую вычисления над парами десятичных чисел и выводящую результат на стандартный файл вывода.

Метод решения

Создадим класс длинных чисел, предавленных ввиде вектора целых чисел. Основанием системы исчисления будем считать 10^9

В каждом элементе вектора хранится цифра, являющаяся числом, меньшим 10^9. Разряды хранятся слева направо (т.е. в 0-ом элементе нулевой разряд, в 1-ом первый и т.д). Операции реализуем, оглядываясь на обычные школьные операции столбиком.

Описание программы

Класс длинных чисел, помимо методов для арифметических действий, содержит следующие поля: вектор целых чисел, в котором хранится само число, основание системы счисления, кол-во цифр в основании (нужно для ввода/вывода), и флаг для отслеживания некорректных операций (деление на ноль). Особые методы - normalize и shift_right. первый убирает незначащие нули из вектора, второй сдвигает число вправо (т.е. в обычной записи числа добавляем справа 0, увеличивая кол-во его разрядов на 1)

Метод	Значение
operator-=	оператор декременции, вычитает из левого операнда правый, перезаписывает левый
operator*(uul& left, uul& right)	умножение двух длинных чисел. Возвращает новое длинное число
operator*=	умножение длинных, модифицирует левый операнд
operator*(uul& left, int right)	умножение длинного на обычное целое число.

Дневник отладки

При создании этой таблицы была использована история посылок.

Время	ведрикт	Описание
2020/09/29 12:53:08	Ошибка выполнения	утечка памяти
2020/09/29 12:56:27	Неправильный ответ	Ошибка вывода
2020/10/01 13:02:51	Неправильный ответ	Ошибка в алгоритме возведения в степень
2020/09/14 06:18:32	Ожидает подтверждения	ура

листинг

main.cpp

```
#include <iostream>
#include "unsigned_ultra_long.h"

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);


    std::string num1, num2;
    char c;
    auto long_num3 = new uul;
    while (std::cin >> num1 >> num2 >> c) {
        uul long_num1(num1);
        uul long_num2(num2);
        switch (c) {
            case '+':
                std::cout << (long_num1 += long_num2) << '\n';
                break;
            case '-':
                std::cout << (long_num1 -= long_num2) << '\n';
                break;
            case '*':
                *long_num3 = long_num1 * long_num2;
                std::cout << *long_num3 << '\n';
```

```
                break;
            case '^':
                *long_num3 = long_num1 ^ long_num2;
                std::cout << *long_num3 << '\n';
                break;
            case '/':
                *long_num3 = long_num1 / long_num2;
                std::cout << *long_num3 << '\n';
                break;
            case '>':
                std::cout << (long_num1 > long_num2 ? "true" : "false")
                    << '\n';
                break;
            case '<':
                std::cout << (long_num1 < long_num2 ? "true" : "false")
                    << '\n';
                break;
            case '=':
                std::cout << (long_num1 == long_num2 ? "true" : "false")
                    << '\n';
                break;
            default:
                std::cout << "input error\n";
                return -1;
        }
    }
    delete long_num3;
    return 0;
}
```

unsigned_ultra_long.h

```
#ifndef DA_LAB6_UNSIGNED_ULTRA_LONG_H
#define DA_LAB6_UNSIGNED_ULTRA_LONG_H

#include <iostream>
#include <vector>

class uul {
public:
    uul() = default;
    uul(std::string& s);
```

```
    friend std::ostream& operator<<(std::ostream& os, const uul& num);
    friend uul& operator+=(uul& left, uul& right);
    friend bool operator>(const uul& left, const uul& right);
    friend bool operator<(uul& left, uul& right);
    friend bool operator==(uul& left, uul& right);
    friend bool operator==(uul& left, int right);
    friend uul& operator-=(uul& left, const uul& right);
    friend uul operator*(uul& left, uul& right);
    friend uul& operator*=(uul& left, uul& right);
    friend uul operator*(uul& left, int right);
    friend uul operator^(uul& left, uul& right);
    friend uul operator/(uul& left, uul& right);
    ~uul() = default;
private:
    void shift_right();
    void normalize();
    std::vector<int> digits;
    int base = 1000*1000*1000;
    int digits_in_base = 9;
    bool errFlag = false;
};

#endif //DA_LAB6_UNSIGNED_ULTRA_LONG_H
```

unsigned_ultra_long.cpp

```
#include <sstream>
#include "unsigned_ultra_long.h"

uul::uul(std::string& s) {
    for (long long i = s.length(); i > 0; i -= digits_in_base) {
        if (i < digits_in_base) {
            digits.push_back(atoi(s.substr(0,i).c_str()));
        } else {
            digits.push_back(atoi(
                s.substr(i-digits_in_base,digits_in_base).c_str()
            ));
        }
    }
    normalize();
}
```

```
void uul::normalize() {
    while (digits.size() > 1 && digits.back() == 0) {
        digits.pop_back();
    }
}

std::ostream& operator<<(std::ostream & os, const uul& num) {
    if (num.errFlag) {
        os << "Error";
        return os;
    }
    os << (num.digits.empty() ? 0 : num.digits.back());
    os.fill('0');
    for (long long i = (long long)num.digits.size()-2; i >=0; --i) {
        os.width(num.digits_in_base);
        os << num.digits[i];
    }
    return os;
}

uul& operator+=(uul& left, uul& right) {
    int carry = 0;
    for (size_t i = 0; i < std::max(left.digits.size(), right.digits.size())
    || carry; ++i) {
        if (i == left.digits.size()) left.digits.push_back(0);
        left.digits[i] += carry +
            (i < right.digits.size() ? right.digits[i] : 0);
        carry = left.digits[i] >= left.base;
        if (carry) left.digits[i] -= left.base;
    }
    return left;
}

bool operator>(const uul& left, const uul& right) {
    if (left.digits.size() < right.digits.size()) return false;
    if (left.digits.size() > right.digits.size()) return true;
    for (int i = (int)left.digits.size()-1; i >= 0; --i) {
        if (left.digits[i] > right.digits[i]) return true;
        if (left.digits[i] < right.digits[i]) return false;
    }
    return false;
```

```
}

bool operator<(uul& left, uul& right) {
    if (left.digits.size() > right.digits.size()) return false;
    if (left.digits.size() < right.digits.size()) return true;
    for (int i = (int)left.digits.size()-1; i >= 0; --i) {
        if (left.digits[i] < right.digits[i]) return true;
        if (left.digits[i] > right.digits[i]) return false;
    }
    return false;
}

bool operator==(uul& left, uul& right) {
    if (left.digits.size() != right.digits.size()) return false;
    for (int i = (int)left.digits.size()-1; i >= 0; --i) {
        if (left.digits[i] != right.digits[i]) return false;
    }
    return true;
}

bool operator==(uul& left, int right) {
    std::stringstream ss;
    ss << right;
    std::string str = ss.str();
    uul b(str);
    if (left.digits.size() != b.digits.size()) return false;
    for (int i = (int)left.digits.size()-1; i >= 0; --i) {
        if (left.digits[i] != b.digits[i]) return false;
    }
    return true;
}

uul& operator-=(uul& left, const uul& right) {
    if (right > left) {
        left.errFlag = true;
        return left;
    }
    int carry = 0;
    for (size_t i = 0; i < right.digits.size() || carry; ++i) {
        left.digits[i] -= carry +
        (i < right.digits.size() ? right.digits[i] : 0);
        carry = left.digits[i] < 0;
```

```
        if (carry) left.digits[i] += left.base;
    }
    while (left.digits.size() > 1 && left.digits.back() == 0) {
        left.digits.pop_back();
    }
    return left;
}

uul operator*(uul& left, uul& right) {
    uul result;
    result.digits.resize(left.digits.size() + right.digits.size());

    for (size_t i = 0; i < left.digits.size(); ++i) {
        int carry = 0;
        for (size_t j = 0; j < right.digits.size() || carry != 0; ++j) {
            long long tmp = result.digits[i + j] + carry +
                (long long)left.digits[i] *
                (long long)(j < right.digits.size() ? right.digits[j] : 0);
            carry = static_cast<int>(tmp / left.base);
            result.digits[i+j] = static_cast<int>(tmp % left.base);
        }
    }
    while (result.digits.size() > 1 && result.digits.back() == 0) {
        result.digits.pop_back();
    }
    return result;
}

uul& operator*=(uul& left, uul& right) {
    auto result = new uul;
    result->digits.resize(left.digits.size() + right.digits.size());
    for (size_t i = 0; i < left.digits.size(); ++i) {
        int carry = 0;
        for (size_t j = 0; j < right.digits.size() || carry != 0; ++j) {
            long long tmp = result->digits[i + j] + carry +
                (long long)left.digits[i] *
                (long long)(j < right.digits.size() ? right.digits[j] : 0);
            carry = static_cast<int>(tmp / left.base);
            result->digits[i+j] = static_cast<int>(tmp % left.base);
        }
    }
    while (result->digits.size() > 1 && result->digits.back() == 0) {
```

```
        result->digits.pop_back();
    }
    left = *result;
    delete result;
    left.normalize();
    return left;
}

uul operator*(uul& left, int right) {
    uul result;
    result.digits.resize(left.digits.size() + (right / left.base) + 1);
    int carry = 0;
    for (size_t i = 0; i < left.digits.size() || carry != 0; ++i) {
        long long tmp = carry +
            (long long) ( i < left.digits.size() ? left.digits[i] : 0) *
            (long long)right;
        carry = static_cast<int>(tmp / left.base);
        result.digits[i] = static_cast<int>(tmp % left.base);
    }
    result.normalize();
    return result;
}

void uul::shift_right() {
    if (digits.empty()) {
        digits.push_back(0);
        return;
    }
    digits.push_back(digits[digits.size()-1]);
    for (size_t i = digits.size() - 2; i > 0; --i)
        digits[i] = digits[i-1];
    digits[0] = 0;
}

uul operator/(uul& left, uul& right) {
    uul result;
    uul current;
    if (right == 0) {
        result.errFlag = true;
        return result;
    }
    result.digits.resize(left.digits.size());
```

```
    for (long long i =
    static_cast<long long>(left.digits.size())-1; i >= 0; --i) {
        current.shift_right();
        current.digits[0] = left.digits[i];
        current.normalize();
        int x = 0, l = 0, r = current.base;
        while (l <= r) {
            int m = (l + r) / 2;
            uul t = right * m;
            if (!(t > current)) {
                x = m;
                l = m + 1;
            } else r = m - 1;
        }
        result.digits[i] = x;
        current -= (right * x);
    }
    result.normalize();
    return result;
}

uul operator^(uul& left, uul& right) {
    uul res;
    if (left == 0 && right == 0) {
        res.errFlag=true;
        return res;
    }
    std::string _one("1");
    uul one(_one);
    std::string _two("2");
    uul two(_two);
    res.digits.push_back(1);
    while (!(right == 0)) {
        if (right.digits[0] % 2 == 1) {
            res *= left;
            right -= one;
        } else {
            left *= left;
            right = right / two;
        }
    }
    res.normalize();
```

```
    return res;
}
```

Недочёты

Реализация операторов довольно хаотична: для некоторых операций есть два оператора - перезаписывающий и обычный, а для возведения в степень нет.

Выводы

Проделав данную работу, я изучил различные методы реализации длинной арифметики, зачем это вообще нужно. Как бонус - узнал, почему в Python медленное умножение (слышал это довольно давно, только после этой работы понял, о чём вообще речь).