Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу
«Дискретный анализ»

Студент: Ибрагимов Р. Р.
Группа: М8О-303Б-22
Преподаватель: Макаров Н.К.
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Постановка задачи

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.

Список арифметических операций:

- Сложение (+).
- Вычитание (-).
- Умножение (*).
- Возведение в степень (^).
- Деление (/).

Замечание: при реализации деления можно ограничить делитель цифрой внтуреннего представления «длинных» чисел, в этом случае максимальная оценка, которую можно получить за лабораторную работу, будет ограничена оценкой 3 («удовлетворительно»).

В случае возникновения переполнения в результате вычислений, попытки вычесть из меньшего числа большее, деления на ноль или возведении нуля в нулевую степень, программа должна вывести на экран строку Error.

Список условий:

- Больше (>).
- Меньше (<).
- Равно (=).

В случае выполнения условия, программа должна вывести на экран строку true, в противном случае – false.

Формат ввода

Входный файл состоит из последовательностей заданий, каждое задание состоит их трех строк:

- Первый операнд операции.
- Второй операнд операции.
- Символ арифметической операции или проверки условия (+, -, *, ^, /, >, <, =).

Числа, поступающие на вход программе, могут иметь «ведущие нули».

Формат вывода
Для каждого задания из входного файла нужно распечатать результат на отдельной строке в выходном файле:

- Числовой результат для арифметических операций.
- Строку Error в случае возникновения ошибки при выполнении арифметичесокй операции.
- Строку true или false при выполнении проверки условия.

В выходных данных вывод чисел должен быть нормализован, то есть не содержать в себе «ведущих» нулей.

Метод решения
Будем представлять число в виде вектора digits, где каждый элемент которого — блок из DIGIT_LENGH цифр числа в обратном порядке. Например, число 123456789 может быть представлено как [6789, 2345, 1].
Для ввода нужно уметь преобразовать число, записанное с помощью строки, в массив цифр, при этом не учитывая незначащие нули. Для вывода нужно производить обратное действие, при этом правильно выводить значащие нули.
Сложение и вычитание реализуем наиболее простым способ — в столбик. Их сложность O(max(n,m)), где n — количество разрядов первого числа, m — количество разрядов второго числа (в случае вычитания в качестве первого числа всегда выступает большее число).
Существуют различные быстрые алгоритмы умножения, но часто хватает и базового умножения в столбик. Его и реализуем. Сложность O(n * m), что приблизительно равняется O(n^2).
К делению надо подходить аккуратно, потому что при делении в столбик требуется быстро определять наибольшее число, на которое можно разделить текущий остаток на делитель. Поэтому для эффективной реализации будем использовать бинарный поиск такого числа. Оценка сложности составляет O(n * m * log_2(MAX_DIGIT)).
Для возведения числа в степень воспользуемся алгоритмом бинарного возведения, который основывается на представлении показателя степени как суммы степеней двойки. Итоговую сложность можно оценить как O(n^2*p^2*log_2(p)), где p — показатель степени.
Операции сравнения (<, =, >) весьма элементарны, они осуществляются поэлементно внутреннего представления чисел. Сложность O(max(n,m)).

Исходный код
```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>
#include <numeric>

class BigInt {
private:
    const static int DIGIT_LENGTH = 4;
```

```
    const static int MAX_DIGIT = 1e4;

    std::vector<int> digits;

    void RemoveLeadingZeros();

public:
    BigInt();
    BigInt(const size_t& size);
    BigInt(const std::string& str);

    BigInt operator+(const BigInt& other) const;
    BigInt operator-(const BigInt& other) const;
    BigInt operator*(const BigInt& other) const;
    BigInt operator^(const BigInt& other) const;
    BigInt operator/(const BigInt& other) const;

    bool operator<(const BigInt& other) const;
    bool operator>(const BigInt& other) const;
    bool operator==(const BigInt& other) const;
    bool operator<=(const BigInt& other) const;
    bool operator>=(const BigInt& other) const;

    friend std::ostream& operator<< (std::ostream& os, const
BigInt& number);
    friend std::istream& operator>> (std::istream& is, BigInt&
number);
};

BigInt::BigInt() : digits(1, 0) {};

BigInt::BigInt(const size_t& size) : digits(size, 0) {}

BigInt::BigInt(const std::string& str) {
    size_t start = str.size();
    while (start > 0) {
        size_t len = std::min(static_cast<size_t>(DIGIT_LENGTH),
start);
        digits.push_back(std::stoi(str.substr(start - len, len)));
        start -= len;
    }
}

void BigInt::RemoveLeadingZeros() {
    while (digits.size() > 1 && digits.back() == 0) {
        digits.pop_back();
    }
}

BigInt BigInt::operator+(const BigInt& other) const {
    BigInt result(std::max(digits.size(), other.digits.size()) +
1);
```

```
    int carry = 0;

    for (size_t i = 0; i < result.digits.size() || carry; ++i) {
        int this_digit = i < digits.size() ? digits[i] : 0;
        int other_digit = i < other.digits.size() ?
other.digits[i] : 0;

        int sum = this_digit + other_digit + carry;
        carry = sum / MAX_DIGIT;
        result.digits[i] = sum % MAX_DIGIT;
    }

    result.RemoveLeadingZeros();
    return result;
}

BigInt BigInt::operator-(const BigInt& other) const {
    BigInt result(digits.size());
    int borrow = 0;

    for (size_t i = 0; i < result.digits.size(); ++i) {
        int this_digit = digits[i];
        int other_digit = i < other.digits.size() ?
other.digits[i] : 0;

        int diff = this_digit - other_digit - borrow;

        if (diff < 0) {
            diff += MAX_DIGIT;
            borrow = 1;
        }
        else {
            borrow = 0;
        }
        result.digits[i] = diff;
    }

    result.RemoveLeadingZeros();
    return result;
}

BigInt BigInt::operator*(const BigInt& other) const {
    BigInt result(digits.size() + other.digits.size());

    for (size_t i = 0; i < digits.size(); ++i) {
        long long carry = 0;
        for (size_t j = 0; j < other.digits.size() || carry; ++j)
{
            long long product = result.digits[i + j] +
                                carry +
                                static_cast<long long>(digits[i])
* (j < other.digits.size() ? other.digits[j] : 0);
```

```
            result.digits[i + j] = product % MAX_DIGIT;
            carry = product / MAX_DIGIT;
        }
    }

    result.RemoveLeadingZeros();
    return result;
}

BigInt BigInt::operator^(const BigInt& other) const {
    BigInt result("1");
    BigInt base = *this;
    BigInt exp = other;

    while (exp > BigInt("0")) {
        if ((exp.digits[0] % 2) == 1) {
            result = result * base;
        }
        base = base * base;
        exp = exp / BigInt("2");
    }

    result.RemoveLeadingZeros();
    return result;
}

BigInt BigInt::operator/(const BigInt &other) const {
    BigInt dividend = *this;
    BigInt divisor = other;
    BigInt result(dividend.digits.size());
    BigInt current;

    for (size_t i = dividend.digits.size(); i > 0; --i) {
        current.digits.insert(current.digits.begin(),
dividend.digits[i - 1]);
        current.RemoveLeadingZeros();

        int low = 0, high = BigInt::MAX_DIGIT - 1;
        int quotient = 0;
        while (low <= high) {
            int mid = std::midpoint(low, high);
            BigInt product = divisor *
BigInt(std::to_string(mid));
            if (product <= current) {
                quotient = mid;
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
```

```
        current = current - (divisor *
BigInt(std::to_string(quotient)));

        result.digits[i - 1] = quotient;
    }

    result.RemoveLeadingZeros();
    return result;
}


bool BigInt::operator<(const BigInt& other) const{
    if (digits.size() != other.digits.size()) {
        return digits.size() < other.digits.size();
    }

    for (size_t i = digits.size(); i > 0; --i) {
        if (digits[i - 1] != other.digits[i - 1]) {
            return digits[i - 1] < other.digits[i - 1];
        }
    }

    return false;
}

bool BigInt::operator>(const BigInt& other) const{
    return other < *this;
}

bool BigInt::operator==(const BigInt& other) const{
    return digits == other.digits;
}

bool BigInt::operator<=(const BigInt& other) const{
    return !(*this > other);
}

bool BigInt::operator>=(const BigInt& other) const{
    return !(*this < other);
}

std::ostream& operator<< (std::ostream& os, const BigInt& number)
{
     if (number.digits.empty()) {
        os << "0";
        return os;
    }

    os << number.digits.back();

    for (size_t i = number.digits.size() - 1; i > 0; --i) {
```

```
        os << std::setw(BigInt::DIGIT_LENGTH) << std::setfill('0')
<< number.digits[i - 1];
    }

    return os;
}

std::istream& operator>> (std::istream& is, BigInt& number) {
    std::string input;
    is >> input;

    number = BigInt(input);
    number.RemoveLeadingZeros();

    return is;
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    std::cout.tie(0);

    BigInt number1, number2;
    char operation;

    while (std::cin >> number1 >> number2 >> operation) {

        if      (operation == '+') {
                BigInt res = number1 + number2;
                std::cout << res << '\n';
        }
        else if (operation == '-') {
            if (number1 < number2) {std::cout << "Error\n";}
            else {
                BigInt res = number1 - number2;
                std::cout << res << '\n';
            }
        }
        else if (operation == '*') {
                BigInt res = number1 * number2;
                std::cout << res << '\n';
        }
        else if (operation == '^') {
            if (number1 == BigInt("0") and number2 == BigInt("0"))
{std::cout << "Error\n";}
            else{std::cout << (number1 ^ number2) << '\n';}
        }
        else if (operation == '/') {
            if (number2 == BigInt("0")) {std::cout << "Error\n";}
            else {
                BigInt res = number1 / number2;
                std::cout << res << '\n';
```

```
            }
        }
        else if (operation == '<') {
            if (number1 < number2)  {std::cout << "true\n";} else
{std::cout << "false\n";}
        }
        else if (operation == '>') {
            if (number1 > number2)  {std::cout << "true\n";} else
{std::cout << "false\n";}
        }
        else if (operation == '=') {
            if (number1 == number2) {std::cout << "true\n";} else
{std::cout << "false\n";}
        }
    }

    return 0;
}
```

Тесты

Номер теста	Ввод	Вывод
1	123 456 + 14667788 1344 + 4200000000000 17000001 + 12345600000000000000000000000000000000000000000000000000000000000000000000000000 654321 + 12345678900000 0 +	579 14669132 4200017000001 12345600000000000000000000000000000000000000000000000000000000000000000000000000 12345678900000
2	15345 45 - 12 12467 - 132435465768776539999999999	15300 Error 132435465768776539823001516 1 0 12345678900000

	35769984838 - 145 145 - 12345678900000 0 -	
3	2 5 * 1234586878 1 * 145466787678 0 * 123456789 987654321 * 1234500000000000 14 * 123450 10000 *	10 1234586878 0 121932631112635269 17283000000000000 1234500000
4	10 5 / 1345678995455 0 / 426868 2 / 1589225586804 12 /	2 Error 213434 132435465567
5	3 0 ^ 5 2	1 25 Error 18446744073709551616 126765060022822940149670320537 6

	^ 0 0 ^ 2 64 ^ 2 100 ^ 0 5 ^	0
6	123456 12345678 < 123456 12345678 = 123456 12345678 > 1234 1234 = 1234 132456 + 987654 13245 - 3456 2345 * 24 25 ^ 5640 4 /	true false false true 133690 974409 8104320 32009658644406818986777955348250624 1410

Выводы

В ходе выполнения данной лабораторной работы я изучил, как представлять и эффективно хранить большие числа, значения которых превышают

стандартных целочисленных типов данных. Я реализовал длинную арифметику с основными арифметическими операциями. Также открыл для себя, что существуют различные варианты алгоритмов каких-то операций.
Наиболее сложными оказались операции деления и возведения в степень.
С точки зрения непосредственно программирования я получил удовольствие от написания кода в стиле ООП.