Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский Авиационный Институт»

Национальный Исследовательский Университет

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №6

по курсу «Дискретный анализ»

Студент:	Хренникова А. С.
Группа:	М8О-208Б-19
Преподаватель:	Капралов Н. С.
Подпись:	
Оценка:	
Дата:	

Москва, 2021

Лабораторная работа №6

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(/).

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.

Количество десятичных разрядов целых чисел не превышает 100000. Основание выбранной системы счисления для внутреннего представления «длинных» чисел должно быть не меньше 10000.

Формат входных данных:

Входной файл состоит из последовательности заданий, каждое задание состоит из трех строк:

Первый операнд операции.

Второй операнд операции.

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

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

Формат результата:

Для каждого задания из выходного файла нужно распечатать результат на отдельной строке в выходном файле:

Числовой результат для арифметических операций.

Строку Error в случае возникновения ошибки при выполнении арифметической 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.

Москва, 2021

1 Описание

Длинная арифметика — в вычислительной технике операции (сложение, умножение, вычитание, деление, возведение в степень и т.д.) над числами, разрядность которых превышает длину машинного слова данной вычислительной машины. Эти операции реализуются не аппаратно, а программно, используя базовые аппаратные средства работы с числами меньших порядков.

В общем случае, любое число можно представить в виде:
$A = a_{n-1}\beta^{n-1} + a_{n-2}\beta^{n-2} + \ldots + a_1\beta + a_0$
где β – основание системы счисления, в которой мы представляем число, а коэффициенты a_i удовлетворяют двойному неравенству $0 \leq a_i < \beta$.
Представление числа напоминает представление многочлена, только вместо x в соответствующей степени имеем основание β в нужной степени. Как известно, многочлен $a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ удобно представлять в виде массива, элементы которого представляют коэффициенты a_i, а индекс i — определяет соответствующую степень x. Длинное число хранится аналогично, основание $\beta = 10000$.

Операции над числами высокой точности реализованы на основе простых операций сложения, вычитания, умножения и деления в столбик, которым учат детей в начальной школе. Фактически большинство алгоритмов – не что иное, как механическое воспроизведение знакомых операций, выполняемых при помощи карандаша и бумаги.

Сложность операции сложения: O(max(n, m)), вычитания: O(n), где n, m – количество разрядов в числах (для вычитания n<m), умножения: O(n * m), возведения в степень: O(log(m) * n * n), где n – количество разрядов числа, m – степень возведения.

Москва, 2021

2 Исходный код:

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
#include <string>
#include <cmath>

const int BASE = 10000;
const int DIGIT_LENGTH = 4;

class TBigInt {
private:
   std::vector<int> bigInt;

public:
   TBigInt(int value);
   TBigInt(const std::string& value);
   int GetDigit(int id) const;
   void SetDigit(int position, int value);
   TBigInt operator+(const TBigInt& other) const;
   TBigInt operator-(const TBigInt& other) const;
   TBigInt operator*(const TBigInt& other) const;
   TBigInt operator/(const TBigInt& other) const;
   TBigInt Pow(TBigInt& first);
   void Shift(int sh);
   void DeleteZeros();
   operator unsigned long long();
   friend int Compare(const TBigInt& first, const TBigInt& second);
   bool operator<(const TBigInt& other) const;
   bool operator>(const TBigInt& other) const;
   bool operator==(const TBigInt& other) const;
   friend std::ostream& operator<<(std::ostream& out, TBigInt num);
};

TBigInt::TBigInt(int value) {
   if (value == 0) {
      bigInt.push_back(0);
   } else {
      for (int i = value; i > 0; i /= BASE) {
         bigInt.push_back(i % BASE);
      }
   }
}

TBigInt::TBigInt(const std::string& value) {
   for (int i = value.length() - 1; i >= 0; i -= DIGIT_LENGTH) {
      int curDigit = 0;
      int curDegree = 1;
      for (int j = 0; j < DIGIT_LENGTH && i >= j; ++j) {
         curDigit += (value[i - j] - '0') * curDegree;
         curDegree *= 10;
      }
      bigInt.push_back(curDigit);
   }
```

```
    DeleteZeros();
}

void TBigInt::DeleteZeros() {
    while (bigInt.size() > 1 && bigInt.back() == 0) {
        bigInt.pop_back();
    }
}

int TBigInt::GetDigit(int id) const {
    if (id >= bigInt.size()) {
        return 0;
    }
    if (id < 0) {
        throw std::logic_error("Wrong id");
    }
    return bigInt[id];
}

void TBigInt::SetDigit(int position, int value) {
    if (position > bigInt.size() - 1) {
        bigInt.resize(position + 1);
    }
    bigInt[position] = value;
}

TBigInt TBigInt::operator+(const TBigInt& other) const {
    int newSize = std::max(bigInt.size(), other.bigInt.size()) + 1;
    TBigInt res(0);
    res.bigInt.resize(newSize);
    int tmp;
    int k = 0;
    for (int i = 0; i < newSize - 1; ++i) {
        tmp = GetDigit(i) + other.GetDigit(i) + k;
        k = tmp / BASE;
        res.SetDigit(i, tmp % BASE);

    }
    if (k != 0) {
        res.SetDigit(newSize - 1, k);
    }
    res.DeleteZeros();
    return res;
}

TBigInt TBigInt::operator-(const TBigInt& other) const {
    if (*this < other) {
        throw std::logic_error("Error");
    }
    int newSize = bigInt.size();
    TBigInt res(0);
    res.bigInt.resize(newSize);
    int k = 0;
    int tmp;
    for (int i = 0; i < newSize; ++i) {
        tmp = GetDigit(i) - other.GetDigit(i) - k;
```

```
        k = (tmp < 0);
        if (k) {
            tmp += BASE;
        }
        res.SetDigit(i, tmp);
    }
    res.DeleteZeros();
    return res;
}

TBigInt TBigInt::operator*(const TBigInt& other) const {
    int newSize = bigInt.size() + other.bigInt.size();
    TBigInt res(0);
    res.bigInt.resize(newSize);
    for (int j = 0; j < other.bigInt.size(); ++j) {
        if (other.bigInt[j] == 0) {
            continue;
        }
        int k = 0;
        for (int i = 0; i < bigInt.size(); ++i) {
            int tmp = GetDigit(i) * other.GetDigit(j) + k + res.GetDigit(i + j);
            k = tmp / BASE;
            res.SetDigit(i + j, tmp % BASE);
        }
        if (k) {
            res.SetDigit(j + bigInt.size(), k);
        }
    }
    res.DeleteZeros();
    return res;
}

TBigInt TBigInt::operator/(const TBigInt& other) const {
    TBigInt res(0);
    if (other.GetDigit(0) == 0 && other.bigInt.size() == 1) {
        throw std::logic_error("Error");
    }
    int n = other.bigInt.size();
    if (*this < other) {
        return res;
    }
    int m = bigInt.size() - n;
    int d = BASE / (other.GetDigit(n - 1) + 1);
    TBigInt u = *this;
    std::string s1 = std::to_string(d);
    TBigInt tmp1(s1);
    u = u * tmp1;
    u.bigInt.push_back(0);
    TBigInt v = other;
    v = v * tmp1;
    res.bigInt.resize(m + 1);
    TBigInt qv(0);
    unsigned long long q;
    unsigned long long r;
    for (int j = m; j >= 0; j--) {
        q = ((unsigned long long)u.GetDigit(j + n) * BASE + u.GetDigit(n + j - 1)) / v.GetDigit(n - 1);
```

```
        r = ((unsigned long long)u.GetDigit(j + n) * BASE + u.GetDigit(n + j - 1)) - v.GetDigit(n - 1) * q;
        while ((q == BASE || (q * v.GetDigit(n - 2)) > (BASE * r + u.GetDigit(j + n - 2))) && r < BASE) {
            q -= 1;
            r += v.GetDigit(n - 1);
        }
        qv = v;
        std::string s2 = std::to_string(q);
        TBigInt tmp2(s2);
        qv = qv * tmp2;
        qv.Shift(j);
        if (u < qv) {
            u.bigInt.push_back(1);
            q -= 1;
        }
        u = u - qv;
        res.SetDigit(j, q);
    }
    while (res.bigInt.size() > 1 && res.bigInt.back() == 0) {
        res.bigInt.pop_back();
    }
    res.DeleteZeros();
    return res;
}

TBigInt::operator unsigned long long() {
    unsigned long long out = 0;
    for (int i = 0; i < bigInt.size(); i++) {
        out += (unsigned long long)GetDigit(i) * (unsigned long long)pow(BASE, i);
    }
    return out;
}

void TBigInt::Shift(int sh) {
    if (bigInt.size() == 1 && GetDigit(0) == 0) {
        return;
    }
    TBigInt new_num(0);
    new_num.bigInt.resize(bigInt.size() + sh, 0);
    copy(bigInt.begin(), bigInt.end(), new_num.bigInt.begin() + sh);
    std::swap(new_num, *this);
}

TBigInt TBigInt::Pow(TBigInt& other) {
    TBigInt res(1), zero(0), one(1), two(2);
    if (*this == zero && other == zero) {
        throw std::logic_error("Error");
    }
    while (other > zero) {
        if (other.GetDigit(0) % 2 == 1) {
            res = res * (*this);
            other = other - one;
        } else {
            (*this) = (*this) * (*this);
            other = other / two;
        }
    }
```

```
    return res;
}

int Compare(const TBigInt& first, const TBigInt& second) {
    if (first.bigInt.size() < second.bigInt.size()) {
        return 1;
    }
    else if (first.bigInt.size() > second.bigInt.size()) {
        return -1;
    }
    for (size_t i = first.bigInt.size(); i > 0; i--) {
        if (first.bigInt[i - 1] < second.bigInt[i - 1]) {
            return 1;
        }
        else if (first.bigInt[i - 1] > second.bigInt[i - 1]) {
            return -1;
        }
    }
    return 0;
}

bool TBigInt::operator<(const TBigInt& other) const {
    return (Compare(*this, other) == 1 ? true : false);
}

bool TBigInt::operator>(const TBigInt& other) const {
    return (Compare(*this, other) == -1 ? true : false);
}

bool TBigInt::operator==(const TBigInt& other) const {
    return (Compare(*this, other) == 0 ? true : false);
}

std::ostream& operator<<(std::ostream& out, TBigInt num) {
    if (num.bigInt.size() == 0) {
        out << 0;
    } else {
        out << num.GetDigit(num.bigInt.size() - 1);
        for (int i = num.bigInt.size() - 2; i >= 0; --i) {
            out << std::setfill('0') << std::setw(DIGIT_LENGTH) << num.GetDigit(i);
        }
    }
    return out;
}

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(nullptr);
    TBigInt zero(0), one(1);
    std::string firstOp, secondOp;
    char s;
    while (std::cin >> firstOp >> secondOp >> s) {
        TBigInt first(firstOp);
        TBigInt second(secondOp);
        if (s == '+') {
            std::cout << first + second << "\n";
```

```
        } else if (s == '-') {
            try {
                std::cout << first - second << "\n";
            }
            catch (std::logic_error& e) {
                std::cout << e.what() << "\n";
            }
        } else if (s == '*') {
            std::cout << first * second << "\n";
        } else if (s == '/') {
            try {
                std::cout << first / second << "\n";
            }
            catch (std::logic_error& e) {
                std::cout << e.what() << "\n";
            }
        } else if (s == '^') {
            try {
                std::cout << first.Pow(second) << "\n";
            }
            catch (std::logic_error& e) {
                std::cout << e.what() << "\n";
            }
        } else if (s == '<') {
            std::cout << (first < second ? "true" : "false") << "\n";
        } else if (s == '>') {
            std::cout << (first > second ? "true" : "false") << "\n";
        } else if (s == '=') {
            std::cout << (first == second ? "true" : "false") << "\n";
        } else {
            std::cout << "Error\n";
        }
    }
}
```

3 Консоль:

```
lina_tucha@LAPTOP-44CRFC1U:~/labs/da$ ./x06
38943432983521435346436
354353254328383
+
90409438473849324729384738 43
2343543
-
972323
2173937
>
2
3
-
3894343333787468967 4819
90409438473849324729361303 00
false
Error
```

4 Тест производительности:

	My(sec.)	Gmp(sec.)
Сложение/вычитание (10^4)	0.1275	0.3510
Умножение(10^4)	2.1191	0.9864
Деление(10^4)	3.3090	1.0046

Очевидно, что питон более оптимален для работы с большими числами.

Москва, 2021

5 Выводы:

Выполняя данную лабораторную работу, я познакомилась с длинной арифметикой, реализовала свой класс для работы с длинными числами.

Москва, 2021