

Московский авиационный институт  
(национальный исследовательский университет)

Институт информационных технологий и прикладной  
математики

Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу «Дискретный анализ»

Студент: Л. В. Ивенкова  
Преподаватель: Н. С. Капралов  
Группа: М8О-208Б-19  
Дата:  
Оценка:  
Подпись:

Москва, 2021

## Лабораторная работа №6

**Задача:** Калькулятор

### Вариант №4

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 ( $\times$ ).

Возведение в степень ( $\wedge$ ).

Деление (/).

В случае возникновения переполнения в результате вычислений, попытки вычесть из меньшего числа большее, деления на ноль или возведения 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 (+, -,  $\times$ ,  $\wedge$ , /, >, <, =).

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.

Строки true или false при выполнении проверки условия.

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

# 1 Описание

Требуется написать реализацию алгоритма работы с длинными числами.

Справка [1]: «**Длинная арифметика** — набор алгоритмов для поразрядной работы с числами произвольной длины. Она применяется как с относительно небольшими числами, превышающими ограничения типа *long long* в несколько раз, так и с по-настоящему большими числами (чаще всего до  $10^{100000}$ ).

Для работы с “длинными” числами их разбивают на разряды. Размер разряда может быть произвольным, но одним из наиболее часто употребляемых вариантов является  $10^4$  — наибольшая степень десяти, квадрат которой не превышает ограничения типа *int*. Он используется для максимальной эффективности при хранении разрядов как чисел типа *int*.

(Ограничения на квадрат размера разряда связаны с необходимостью перемножать между собой разряды. Если квадрат разряда превышает ограничение своего типа, при умножении возможны переполнения.)

В большинстве реализаций разряды хранятся в порядке, обратным привычному для упрощения работы с ними. Например число 578002300 при размере разряда  $10^4$  представляется следующим массивом: {2300, 7800, 5}»

Для хранения неограниченного количества разрядов будем использовать вектор.

## 2 Исходный код

Создадим класс больших чисел TBigInteger. Он будет включать в себя вектор рядов числа digits и методы класса - операнды  $+$ ,  $-$ ,  $\times$ ,  $\wedge$ ,  $/$ ,  $>$ ,  $<$ ,  $=$ ,  $<<$  и вспомогательные функции для них.

Для возведения числа в степень используем алгоритм бинарного возведения в степень, который работает за  $O(\log(deg) * n^2)$ .

Сам код:

### BigInteger.hpp

```
1 | #include <iostream>
2 | #include <iomanip>
3 | #include <vector>
4 | #include <cmath>
5 |
6 | using namespace std;
7 |
8 | const int DIGIT_LENGTH = 4;
9 | const int BASE = 10000;
10 |
11 | class TBigInteger {
12 | private:
13 |     std::vector<int> digits;
14 |     void RemoveLeadingZeros();
15 |
16 | public:
17 |     TBigInteger(int value = 0);
18 |     TBigInteger(const std::string &value);
19 |     int GetDigit(size_t id) const;
20 |     bool operator<(TBigInteger &other) const;
21 |     bool operator==(TBigInteger &other) const;
22 |     bool operator>(TBigInteger &other) const;
23 |     TBigInteger operator+(const TBigInteger &other);
24 |     TBigInteger operator-(const TBigInteger &other);
25 |     TBigInteger operator*(const TBigInteger &other) const;
26 |     TBigInteger operator/(const TBigInteger &other) const;
27 |     TBigInteger operator^(TBigInteger& other);
28 |     friend std::ostream &operator<<(std::ostream &os, TBigInteger bi);
29 | };
```

### BigInteger.cpp

```
1 | #include "BigInteger.hpp"
2 |
3 | TBigInteger ZERO("0");
```

```

4 TBigInteger ONE("1");
5 TBigInteger TWO("2");
6
7 TBigInteger::TBigInteger(int value) {
8     if (value == 0) {
9         digits.push_back(0);
10    }
11    else {
12        for (int i = value; i > 0; i /= BASE) {
13            digits.push_back(i % BASE);
14        }
15    }
16 }
17
18 TBigInteger::TBigInteger(const std::string &value) {
19     for (int i = value.length(); i > 0; i -= DIGIT_LENGTH) {
20         if(i < DIGIT_LENGTH) {
21             digits.push_back(atoi(value.substr(0, i).c_str()));
22         }
23         else {
24             digits.push_back(atoi(value.substr(i - DIGIT_LENGTH, DIGIT_LENGTH).c_str())
25                               );
26         }
27     }
28     RemoveLeadingZeros();
29 }
30 void TBigInteger::RemoveLeadingZeros() {
31     while (digits.size() > 1 && digits.back() == 0) {
32         digits.pop_back();
33     }
34 }
35
36 int TBigInteger::GetDigit(size_t id) const {
37     if (id >= digits.size()) {
38         return 0;
39     }
40     return digits[id];
41 }
42
43 bool TBigInteger::operator<(TBigInteger& other) const {
44     if (digits.size() != other.digits.size()) {
45         return digits.size() < other.digits.size();
46     }
47     for (int i = digits.size() - 1; i > (-1); --i) {
48         if (digits[i] != other.digits[i]) {
49             return digits[i] < other.digits[i];
50         }
51     }

```

```

52     return false;
53 }
54
55 bool TBigInteger::operator==(TBigInteger& other) const {
56     if (digits.size() != other.digits.size()) {
57         return false;
58     }
59
60     for (size_t i = 0; i < digits.size(); ++i) {
61         if (digits[i] != other.digits[i]) {
62             return false;
63         }
64     }
65
66     return true;
67 }
68
69 bool TBigInteger::operator>(TBigInteger& other) const {
70     return !(*this < other || *this == other);
71 }
72
73 TBigInteger TBigInteger::operator+(const TBigInteger& other) {
74     int flag = 0;
75     size_t newSize = std::max(digits.size(), other.digits.size());
76     for (size_t i = 0; i < newSize || flag != 0; ++i) {
77         if (i == digits.size()) {
78             digits.push_back(0);
79         }
80         if (i < other.digits.size()) {
81             digits[i] += flag + other.digits[i];
82         }
83         else {
84             digits[i] += flag;
85         }
86         flag = (digits[i] >= BASE);
87         if (flag != 0) {
88             digits[i] -= BASE;
89         }
90     }
91     return *this;
92 }
93
94 TBigInteger TBigInteger::operator-(const TBigInteger& other) {
95     int flag = 0;
96     for (size_t i = 0; i < other.digits.size() || flag != 0; ++i) {
97         if (i < other.digits.size()) {
98             digits[i] -= flag + other.digits[i];
99         }
100     }

```

```

101         else {
102             digits[i] -= flag;
103         }
104         flag = (digits[i] < 0);
105         if (flag != 0) {
106             digits[i] += BASE;
107         }
108     }
109     RemoveLeadingZeros();
110     return *this;
111 }
112
113 TBigInteger TBigInteger::operator*(const TBigInteger& other) const {
114     int newSize = digits.size() + other.digits.size();
115     TBigInteger mult;
116     mult.digits.resize(newSize);
117
118     for (int j = 0; j < other.digits.size(); ++j) {
119         if (other.digits[j] == 0) {
120             continue;
121         }
122         int flag = 0;
123         for (int i = 0; i < digits.size(); ++i) {
124             int tmp = GetDigit(i) * other.digits[j] + flag + mult.digits[i + j];
125             flag = tmp / BASE;
126             mult.digits[i + j] = tmp % BASE;
127         }
128         if (flag) {
129             mult.digits[j + digits.size()] = flag;
130         }
131     }
132     mult.RemoveLeadingZeros();
133     return mult;
134 }
135
136 TBigInteger TBigInteger::operator/(const TBigInteger& other) const {
137     TBigInteger result, curRank(0);
138     result.digits.resize(digits.size());
139
140     for (int i = digits.size() - 1; i >= 0; --i) {
141         curRank.digits.insert(curRank.digits.begin(), digits[i]);
142         curRank.RemoveLeadingZeros();
143
144         int x = 0;
145         int l = 0;
146         int r = BASE;
147         while (l <= r) {
148             int mid = (l + r) / 2;
149             TBigInteger tmp = TBigInteger(mid) * other;

```



```

150         if (!(tmp > curRank)) {
151             x = mid;
152             l = mid + 1;
153         } else {
154             r = mid - 1;
155         }
156     }
157     result.digits[i] = x;
158     curRank = curRank - other * TBigInteger(x);
159 }
160 result.RemoveLeadingZeros();
161 return result;
162 }
163
164 TBigInteger TBigInteger::operator ^(TBigInteger& deg) {
165     TBigInteger res(1);
166     while (deg > ZERO) {
167         if (deg.GetDigit(0) % 2 == 1) {
168             res = res * (*this);
169             deg = deg - ONE;
170         } else {
171             (*this) = (*this) * (*this);
172             deg = deg / TWO;
173         }
174     }
175     return res;
176 }
177
178 std::ostream &operator<<(std::ostream &os, TBigInteger bi) {
179     os << bi.digits.back();
180     for (int i = bi.digits.size() - 2; i >= 0; --i) {
181         std::cout << std::setfill('0') << std::setw(DIGIT_LENGTH) << bi.digits[i];
182     }
183     return os;
184 }

```

## main.cpp

```

1  #include "BigInteger.hpp"
2
3  int main() {
4      std::ios_base::sync_with_stdio(false);
5      std::cin.tie(nullptr);
6      std::cout.tie(nullptr);
7
8      TBigInteger ZERO(0);
9
10     std::string l, r;
11     char op;

```

```

12
13 while (std::cin >> l >> r >> op) {
14     TBigInteger left(l);
15     TBigInteger right(r);
16
17     if (op == '+') {
18         std::cout << left + right << "\n";
19     }
20     else if (op == '-') {
21         if (left < right) {
22             std::cout << "Error\n";
23         }
24         else {
25             std::cout << left - right << "\n";
26         }
27     }
28     else if (op == '*') {
29         std::cout << left * right << "\n";
30     }
31     else if (op == '/') {
32         if (right == ZERO) {
33             std::cout << "Error\n";
34         }
35         else {
36             std::cout << left / right << "\n";
37         }
38     }
39     else if (op == '^') {
40         if (left == ZERO && right == ZERO) {
41             std::cout << "Error\n";
42         }
43         else {
44             std::cout << (left ^ right) << "\n";
45         }
46     }
47     else if (op == '<') {
48         std::cout << (left < right ? "true" : "false") << "\n";
49     }
50     else if (op == '>') {
51         std::cout << (left > right ? "true" : "false") << "\n";
52     }
53     else if (op == '=') {
54         std::cout << (left == right ? "true" : "false") << "\n";
55     }
56     else {
57         std::cout << "Error\n";
58     }
59 }
60 }

```

### 3 Консоль

```
parsifal@DESKTOP-3G70RV4:~/DA/Lab6/$ cat test0.txt
123456789101112131415
151413121110987654321
+
151413121110987654321
123456789101112131415
-
123456789101112131415
1514131211
*
123456789101112131415
1514131211
/
123456789101112131415
151413121110987654321
>
123456789101112131415
151413121110987654321
-
parsifal@DESKTOP-3G70RV4:~/DA/Lab6/$ ./solution <test0.txt
274869910212099785736
27956332009875522906
186929777587838512986185093565
81536387470
false
Error
```

## 4 Тест производительности

Операции сравнения выполняются следующим образом: сначала сравниваем размеры массивов `digits` — в зависимости от того, какое число оказалось больше по длине, выводим ответ. Если же они равны, то начинаем пошагово сравнивать каждую цифру массивов, пока не найдём несовпадение. В итоге сложность сравнения —  $O(n)$ , где  $n$  — длина массива `digits`.

Для выполнения операций сложения и вычитания (в столбик) мы должны пройти по всем разрядам двух чисел, учитывая возможность переполнения (на один разряд). Поэтому сложность сложения —  $O(\max(n, m))$ , где  $n$  и  $m$  — длины двух слагаемых, а вычитания —  $O(n)$ , где  $n$  — длина первого числа (в начале мы сравниваем два числа, и, если первое число окажется меньше второго, то программа вернёт ошибку).

Умножение также выполняем в столбик. Его сложность выйдёт  $O(n * m)$ , так как мы перемножает каждую цифру первого числа с каждой цифрой второго.

Деление будем выполнять уголком. Будем по одному разряду брать из делимого, начиная со старшего, и бинарным поиском подбирать разряды частного. Сложность выйдет  $O(n * (n + \log(BASE) * m))$ , где  $n$  и  $m$  — длины первого и второго чисел, а  $BASE$  — основание системы счисления (в нашем случае  $10^4$ ).

Как уже говорилось в начале, возведение в степень будем выполнять с помощью бинарного возведения в степень со сложностью  $O(\log(deg) * n^2)$ , где  $deg$  — показатель степени, а  $n$  — длина числа.

Сравним работу нашего класса больших чисел с библиотекой GNU MP. ("GMP — это свободная библиотека для производства различных арифметических действий над целыми, рациональными и действительными числами. Разрядность чисел, с которыми работает библиотека ограничивается памятью самой машины"[2]).

Количество строк в тесте	Длина чисел в тесте	Мой класс, мкс	GNU MP, мкс
Тесты сложения и вычитания			
$10^3$	$10^5$	23.2	15.7
$10^5$	$10^2$	2469.7	919.7
Тесты умножения			
$10^3$	$10^5$	93.1	21.6
$10^5$	$10^2$	4915	927.9
$10^5$	$10^2$	2469.7	919.7
Тесты деления			
$10^3$	$10^5$	2940.8	56
$10^5$	$10^2$	127391	2626.3

Как видно, библиотека GNU MP везде выигрывает по времени — так как она использует более оптимизированные алгоритмы: например при делении она намного быстрее находит частное, а при умножении использует алгоритм Карацубы, имеющий сложность  $O(n * \sqrt{n})$ .

## 5 Выводы

Выполнив шестую лабораторную работу по курсу «Дискретный анализ», я реализовала свой класс для работы с большими числами (длинной арифметикой), что было полезным опытом, так как в жизни часто возникает необходимость работать с числами, намного превосходящими стандартные `int` и `long long`.

Основная суть длинной арифметики состоит в том, что мы представляем число в виде массива, хранящего разряды числа в выбранной нами системе счисления — в данном случае я выбрала систему счисления  $10^4$  (наибольшую степень десяти, квадрат которой не превышает ограничения типа `int`).

Данный класс поддерживает все те же операции, что и встроенные типы, но работает с ними намного дольше.

Также я познакомилась с представлением длинной арифметики в библиотеке GNU MP (написанной на языке C) и сравнила её со своим классом. Так как она использует более оптимизированные алгоритмы, то она оказалась намного быстрее моей программы.

## Список литературы

- [1] *Длинная арифметика — brestprog.*  
URL: <https://brestprog.by/topics/longarithmetics/> (дата обращения: 29.08.2021).
- [2] *GNU Multi-Precision Libraryx — Википедия.*  
URL: [https://ru.wikipedia.org/wiki/GNU\\_Multi-Precision\\_Library](https://ru.wikipedia.org/wiki/GNU_Multi-Precision_Library) (дата обращения: 29.08.2021).