

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: А. Т. Бахарев  
Преподаватель: А. А. Кухтичев  
Группа: М8О-206Б  
Дата:  
Оценка:  
Подпись:

Москва, 2018

## Лабораторная работа №6

**Задача:** Необходимо разработать программную библиотеку на языке 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 (\*).
- Деление (/).
- Возведение в степень ( $\hat{\phantom{x}}$ ).

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**.

### Ограничения.

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.

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

1. Первый операнд операции.
2. Второй операнд операции.

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

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

### **Формат выходных данных**

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

1. Числовой результат для арифметических операций.
2. Строку Error в случае возникновения ошибки при выполнении арифметической операции.
3. Строку true или false при выполнении проверки условия.

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

# 1 Метод решения

**Длинная арифметика** — это набор программных средств (структуры данных и алгоритмы), которые позволяют работать с числами гораздо больших величин, чем это позволяют стандартные типы данных.

Основная идея заключается в том, что число хранится в виде массива его цифр. Цифры могут храниться в той или иной системе счисления. Если использовать стандартную (при вводе) десятичную систему счисления, то время работы будет существенно большим, так как количество элементов массива будут соответствовать количеству разрядов десятичного числа.

Поэтому стоит хранить числа в степени  $n$  исходной системы счисления как предлагается в задании —  $10^4$ , но можно взять и больше, но всё зависит от типа данных в массиве.

Я использовал беззнаковый тип `std::size_t`, который имеет предел до  $2 * 2^{31} - 1$ .

Числа могут иметь любую систему счисления, которую укажет пользователь библиотеки, поэтому в каждой операции проверяется совместимость двух чисел. Если они несовместимы, то программа завершается с указанием ошибки.

После каждой бинарной операции из результата удаляются все лидирующие нули.

**Замечание:** задание существенно облегчается неотрицательностью чисел.

## 1 Проверка условия.

В этом случае всё довольно просто. Необходимо пройти по  $i$ -ому разряду двух чисел от старшего к младшему и сравнивать элементы, пока не встретятся разные значения в разрядах чисел. Если таких разрядов нет, то числа равны, иначе большим будет число с первым попавшимся большим разрядом.

**Линейная оценка:**  $\Theta(\min\{n, m\})$ , где  $m, n$  — количество разрядов двух чисел.

## 2 Сложение.

Необходимо пройти от младшего разряда к старшему двух чисел и поместить результат сложения по текущему разряду в отдельную переменную, если значение переменной будет больше основания системы счисления, то прибавляем к следующему элементу разряда результата единицу и вычитаем основание из переменной. Процесс продолжается до тех пор, пока итерация не дойдет до самого старшего разряда двух чисел или остаток для следующего разряда будет равен нулю.

**Линейная оценка:**  $\Theta(\max\{n, m\})$ , где  $m, n$  — количество разрядов двух чисел.

### 3 Вычитание.

Перед проведением операции вычитания проводится проверка — первый операнд больше или равен второму, чтобы результирующее число было неотрицательным. Операция происходит аналогично сложению, только при отрицательности сложения двух разрядов берется разряд из следующего разряда первого операнда.

### 4 Умножение.

Как в обычном умножении столбиком необходимо проитерироваться по разрядам от младшего к старшему второго операнда и произвести умножение на первое число.

**Свойство умножения:** ответ будет содержать максимум  $m+n$  разрядов.

**Линейная оценка:**  $\Theta(n * m)$ , где  $m, n$  — количество разрядов двух чисел.

### 5 Возведение в степень.

Данную операцию можно решить перемножением исходного числа «в лоб», а можно воспользоваться быстрым возведением в степень.

**Линейная оценка:**  $\Theta(n * \log d)$ , где  $n$  — количество разрядов числа, а  $d$  — степень числа.

### 6 Деление.

Количество разрядов у частного от деления не превосходит количества разрядов у делимого, поэтому следует формировать ответ со старшего разряда. На каждой итерации имеем текущее значение, которое пытаемся уменьшить на максимально большое количество раз делимым. Можно было бы необходимое количество раз найти циклом от 0 до максимального значения разряда, но можно обойтись более красивым вариантом, а именно — **бинарным поиском**.

Корректность этого утверждения вытекает из того, что рассматриваемая функция, которую можно представить в виде  $b * x$  (где  $b$  — делимое,  $x$  — подбираемое значение) является **монотонно-возрастающей**.

**Линейная оценка:**  $\Theta(n * \log m)$ , где  $m, n$  — количество разрядов двух чисел.

## 2 Описание библиотеки TLongInt.

public	
TLongInt(); TLongInt(std::size_t nRadix, std::size_t tBase);  TLongInt(const std::string & tLine, const std::size_t tBase); void Show(); TLongInt(const TLongInt &otherNum);	Пустой конструктор. Конструктор создает nRadix пустых разрядов числа с основанием tBase. Конструктор, который преобразует строку в большое число с основанием tBase. Печать числа в stdout поток. Конструктор копирования, необходим для присваивания.
TLongInt operator+(const TLongInt &otherNum); TLongInt operator-(const TLongInt &otherNum); TLongInt operator*(const TLongInt &otherNum); TLongInt operator*(const int &otherNum) const;  TLongInt operator^(int degree); TLongInt operator/(const TLongInt &otherNum); bool isNull(); virtual TLongInt()	Бинарный оператор сложения.  Бинарный оператор вычитания. Бинарный оператор умножения.  Бинарный оператор умножения на маленькое число. Бинарный оператор возведения в степень. Бинарный оператор деления.  Проверка является ли число нулём. Деструктор.
private	
std::size_t isLength() const; void SetData(std::size_t position, int row);  int GetData(std::size_t position) const; void ShiftUp(); void DeleteZeros();	Возвращает количество разрядов числа. Меняет значение разряда на row в позиции position. Возвращает значение разряда position. Смещает все разряды вверх (для деления). Удаляет ведущие нули.
protected	
std::vector<int> data; std::size_t length; std::size_t base; std::size_t fullBase;	Вектор разрядов. Количество разрядов. Степень основания. Основание системы счисления.

### 3 Консоль

```
alex$ make
g++ -std=c++11 -o diskran_laba_6 -pedantic -Wall -Werror -Wno-sign-compare -Wno-long-
alex$ ./*6
10 500000000000 +
500000000010
1000 1000 *
1000000
16128735 9712 /
1660
6 10 ^
60466176
2 3 >
false
5 5 =
true
90 1 -
89
alex$
```

## 4 Выводы

Данная лабораторная работа была довольно интересной. Даже операции с числами не бывают простыми и можно легко найти подводный камень. Самое сложное было написать алгоритм деления двух чисел: подбирать бинарным поиском максимальный по системе счисления множитель, на который следует делить первое слагаемое. Метод Карацубы дает выигрыш в умножении, но только с более длинными числами. Если необходимы более быстрые вычисления, то нужно всего лишь явно указать более высокую степень десятичной системы счисления, чем указана по умолчанию. Можно использовать написанную библиотеку для решения своих повседневных задач. Но ведь в Python есть поддержка длинной арифметики. Поэтому пока я не могу найти применение библиотеки, но это был хороший опыт.



## Список литературы

- [1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ, 2-е издание. — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))(дата обращения: 17.12.2018).
- [2] Дональд Кнут, «Искусство программирования», том 2, «Получисленные алгоритмы», 3-е издание. Глава 4.3, «Арифметика многократной точности», стр. 304–335.(дата обращения: 17.12.2018).
- [3] *Алгоритм деления двух чисел*  
URL:<https://mindhalls.ru/big-number-in-c-cpp-add-sub/> (дата обращения: 17.12.2018).
- [4] *Умножение чисел методом Карацубы*  
URL:<https://habrahabr.ru/post/124258/> (дата обращения: 17.12.2018).
- [5] *Перезгрузка операторов*  
URL:<https://habrahabr.ru/post/132014/> (дата обращения: 17.12.2018).