# 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: С. М. Бокоч
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 ($\hat{}$).

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**.

**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 их 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 арифметичесокй 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 указаением ошибки.

После каждой бинарной операции из результата удаляются все лидирующие нули.

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

## 1 Проверка условия.

В этом случае все довольно просто. Неободимо пройтись 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 пройтись 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 TBigNum.

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

# 3 Код дополнительных файлов

К сожалению, я написал генератор тестов после того как отправил задачу на чекер, но он может пригодиться для проверки корректности алгоритмов поиска подстроки в строке данного типа. Также можно использовать мою программу для генерации ответов.

1. checker.py

```
import sys
import random
import string
import copy
from random import choice
count_of_tests = 4
num_requests_in_test = 18

def generate_num(count_rows_in_num):
  return ''.join(choice(num) for i in range(count_rows_in_num))


if __name__=="__main__":
  actions = ["+", "-", "/", "*"]
  condition = [">", "<", "="]
  for enum in range( count_of_tests ):
    step = 2
    count_rows_in_num = 1
    test_file_name = "tests/{:02d}".format( enum + 1 )
    with open( "{0}.t".format( test_file_name ), 'w' ) as output_file, \
      open( "{0}.txt".format( test_file_name ), "w" ) as answer_file:
      for i in range(num_requests_in_test):
        count_rows_in_num *= step
        a = generate_num(count_rows_in_num)
        b = generate_num(count_rows_in_num)
        sign = random.choice(actions[enum])
        output_file.write("{0}\n{1}\n{2}\n".format( a, b, sign ))
        a = int(a)
        b = int(b)
        answer = ""
        if (sign == '+'):
          answer = str(a+b)
        elif sign == '-':
          answer = (str(a-b), "Error")[a < b]
        elif sign == '*':
          answer = str(a*b)
        elif sign == '/':
          answer = str(a//b)
        elif sign == '^':
          answer = str(a**b)
```

```
        elif sign == '<':
          answer = ("false", "true")[a < b]
        elif sign == '>':
          answer = ("false", "true")[a > b]
        elif sign == '=':
          answer = ("false", "true")[a == b]
        answer_file.write("{0}\n".format( answer ))
```

2. makefile

```
FLAGS= -std=c++11 -pedantic -Werror -Wno-sign-compare -Wno-long-long -lm -O2
CC=g++

all: big_int main clear

main: main.cpp
  $(CC) $(FLAGS) -o lab6 main.cpp TBigInt.o

big_int: TBigInt.cpp
  $(CC) $(FLAGS) -c TBigInt.cpp
clear:
  rm -f *.o
  rm -fr *.dSYM
```

3. ./wrapper.sh #Точка входа в «чекер»

```
#!/bin/bash
#clear old tests
rm -rf tests/

mkdir tests
#compile cheker
echo "Compile cheker"
python3 checker.py

#genering program
rm lab6 *.o
make all

echo "Execute tests"
for test_file in `ls tests/*.t`; do
 echo "Execute \${test_file}"
    if ! ./lab6 < \$test_file > tmp ; then
        echo "ERROR"
        continue
    fi
    answer_file="\${test_file\%.*}"

    if ! diff -u "\${answer_file}.txt" tmp > /dev/null ; then
        echo "Failed"
```

```
25      else
26          echo "OK"
27      fi
28  done
29  echo "Finish"
30  rm tmp
```

# 4 Проверка на утечки памяти

Теперь проверим программу на наличие утечек памяти с помощью утилиты **valgrind** на $10^4$ элеметах.

```
[bokoch@MacKenlly lab_2]$ valgrind --leak-check=yes ./main < tests/01.t > dev
==4648==
==4648== HEAP SUMMARY:
==4648==     in use at exit: 72,704 bytes in 1 blocks
==4648==   total heap usage: 110 allocs, 109 frees, 3,728,034 bytes allocated
==4648==
==4648== LEAK SUMMARY:
==4648==    definitely lost: 0 bytes in 0 blocks
==4648==    indirectly lost: 0 bytes in 0 blocks
==4648==      possibly lost: 0 bytes in 0 blocks
==4648==    still reachable: 72,704 bytes in 1 blocks
==4648==         suppressed: 0 bytes in 0 blocks
==4648== Rerun with --leak-check=full to see details of leaked memory
==4648==
==4648== For counts of detected and suppressed errors, rerun with: -v
==4648== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Как видно из протокола, на любом количестве тестов поисходит одна и таже ошибка. Немного разобравшись с кодом утечка была выявлена. И это сама библиотека iostream. Даже при указании данной библиотеки утилита valgrind обнаруживает ошибку.

```
==4599== HEAP SUMMARY:
==4599==     in use at exit: 72,704 bytes in 1 blocks
==4599==   total heap usage: 1 allocs, 0 frees, 72,704 bytes allocated
==4599==
==4599== LEAK SUMMARY:
==4599==    definitely lost: 0 bytes in 0 blocks
```

```
==4599==    indirectly lost: 0 bytes in 0 blocks
==4599==      possibly lost: 0 bytes in 0 blocks
==4599==    still reachable: 72,704 bytes in 1 blocks
==4599==         suppressed: 0 bytes in 0 blocks
==4599== Rerun with --leak-check=full to see details of leaked memory
==4599==
==4599== For counts of detected and suppressed errors, rerun with: -v
==4599== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

**Замечание:** При использовании библиотеки STL, ошибок при написании программы стало гораздо меньше, т.к. не пришлось иметь дело с выделением памяти под динамические массивы, также программу стало легче читать и отлаживать.

# 5 Тест производительности

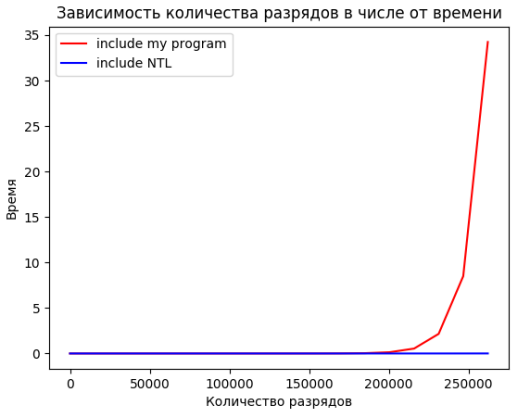
Сравнение производилось с библиотекой **NTL**, которая предназначена для работы с криптографией. Сам листинг программы с использованием библиотеки.

```
#include <NTL/ZZ.h>
#include <NTL/tools.h>
#include <ctime>
#include <cstdio>
#include <iomanip>
using namespace std;
using namespace NTL;
int main(int argc, char *argv[])
{
  std::string lineNum1, lineNum2;
  char sign;

  ZZ a, b, result;

  FILE* fout_multiply;
  FILE* fout_division;
  FILE* fout_addition;
  FILE* fout_subtraction;

  fout_multiply = fopen("benchmark/time/multiply_ntl.txt", "a+");
  fout_division = fopen("benchmark/time/division_ntl.txt", "a+");
  fout_addition = fopen("benchmark/time/addition_ntl.txt", "a+");
  fout_subtraction = fopen("benchmark/time/subtraction_ntl.txt", "a+");

  clock_t start;
  clock_t finish;

  double time;

  while (std::cin >> a >> b >> sign) {
    switch (sign) {
      case '+': {
           start = clock();
               result = a + b;
               finish = clock();
               time = (double) (finish - start)/CLOCKS_PER_SEC;
          fprintf(fout_addition, "%0.15f\n", time);
          std::cout << result << std::endl;
        }
        break;
      case '-':
        if (a < b) {
          std::cout<< "Error\n";
        } else {
```

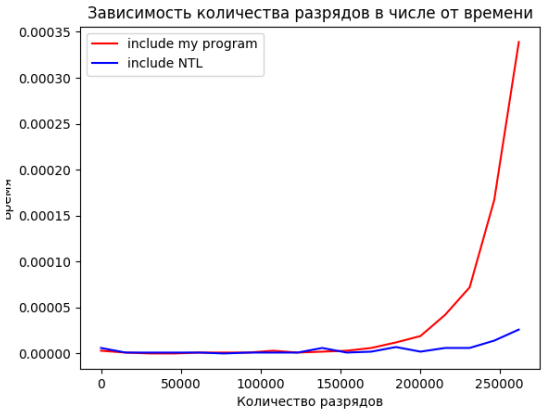
```
45          start = clock();
46                result = a - b;
47                finish = clock();
48                time = (double) (finish - start)/CLOCKS_PER_SEC;
49          fprintf(fout_subtraction, "%0.15f\n", time);
50          std::cout << result << std::endl;
51          break;
52        }
53                break;
54      case '/':
55                if (b == 0) {
56                    std::cout << "Error\n";
57                } else {
58                    start = clock();
59                result = a / b;
60                finish = clock();
61                time = (double) (finish - start)/CLOCKS_PER_SEC;
62          fprintf(fout_division, "%0.10f\n", time);
63          std::cout << result << std::endl;
64                }
65        break;
66      case '*':
67        start = clock();
68                result = a * b;
69                finish = clock();
70                time = (double) (finish - start)/CLOCKS_PER_SEC;
71        fprintf(fout_multiply, "%0.10f\n", time);
72        std::cout << result << std::endl;
73        break;
74      case '^':
75                result = power(a, atoi(lineNum2.c_str()));
76        break;
77      case '>':
78        std::cout << (a > b ? "true" : "false") << std::endl;
79        break;
80      case '<':
81        std::cout << (a < b ? "true" : "false") << std::endl;
82        break;
83      case '=':
84        std::cout << (a == b ? "true" : "false") << std::endl;
85        break;
86      default:
87        std::cout << "Uncorrect requests" << std::endl;
88        break;
89    }
90  }
91  fclose(fout_addition);
92  fclose(fout_subtraction);
93  fclose(fout_multiply);
```

```
94 ||   fclose(fout_division);
95 ||   return 0;
96 || }
```

В результате были получены следующие графики с использованием библиотеки **NumPy** на языке Python.




Умножение




Сложение

Рис. 1: Сравнение скорости работы от количества разрядов.

Замедления работы программы связано с медленной работой стандартного аллокатора. В библиотеке NTL используется *tmalloc*, который заметно ускоряет работоспособность программы.

# 6 Выводы

Данная лабораторная работа была довольно интересной. Даже операции с числами не бывают простыми и можно легко найти подводный камень. Самое сложное было, так это написать алгоритм деления двух чисел: подбирать бинарным поиском максимальный по системе счисления множитель, на который следует делить первое слагаемое. Метод Карацубы дает выйгрыш в умножении, но только с более длинными числами. Если необходимы более быстрые выисления, 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: 10.03.2018).

[2] Дональд Кнут, «Искусство программирования», том 2, «Получисленные алгоритмы», 3-е издание. Глава 4.3, «Арифметика многкратной точности», стр. 304–335.(дата обращения: 18.02.2018).

[3] *Алгоритм деления двух чисел*
URL:`https://mindhalls.ru/big-number-in-c-cpp-add-sub/` (дата обращения: 22.02.2018).

[4] *Умножение чисел методом Карацубы*
URL:`https://habrahabr.ru/post/124258/` (дата обращения: 28.02.2018).

[5] *Перегрузка операторов*
URL:`https://habrahabr.ru/post/132014/` (дата обращения: 29.02.2018).