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: М. С. Гаврилов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №6

Задача: 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.

1 Описание и исходный код

Для реализации длинной арифметики создадим класс TVeryLong. Элементы этого класса будут содержать в себе вектор long long чисел, каждое из которых будет считаться знаком числа в системе счисления с основанием 10^n (n зададутся как константа).

```
1
2  class TVeryLong {
3      public:
4          std::vector<long long> cells;
5          size_t len;
6
7          TVeryLong() {
8              len = 0;
9          }
10
11         TVeryLong(size_t preSize) {
12             len = 0;
13             cells = std::vector<long long>(preSize);
14         }
15
16
17         TVeryLong(std::vector<long long> inp) {
18             //cells = inp;
19             for (size_t i = 0; i < inp.size(); ++i) {
20                 cells.push_back(inp[inp.size() - 1 - i]);
21             }
22             len = inp.size();
23         }
24
25
26         //friend std::ofstream& const operator<< (std::ofstream& out, const my::
               TVeryLong& ptb);
27         friend my::TVeryLong operator+ (const my::TVeryLong& lhs, const my::TVeryLong&
               rhs);
28         friend my::TVeryLong operator- (const my::TVeryLong& lhs, const my::TVeryLong&
               rhs);
29         friend my::TVeryLong operator* (const my::TVeryLong& lhs, const my::TVeryLong&
               rhs);
30         friend my::TVeryLong operator^ (const my::TVeryLong& lhs, const my::TVeryLong&
               rhs);
31         friend my::TVeryLong operator^ (const my::TVeryLong& lhs, const long long& rhs)
               ;
32         friend my::TVeryLong operator/ (const my::TVeryLong& lhs, const my::TVeryLong&
               rhs);
33         friend my::TVeryLong operator/ (const my::TVeryLong& lhs, const long long& rhs)
               ;
34         friend std::ostream& operator<< (std::ostream& out, const my::TVeryLong& ptb);
```

```
35
36          friend bool operator> (const my::TVeryLong& lhs, const my::TVeryLong& rhs);
37          friend bool operator< (const my::TVeryLong& lhs, const my::TVeryLong& rhs);
38          friend bool operator== (const my::TVeryLong& lhs, const my::TVeryLong& rhs);
39      };
```

Сложение будем реализовывать поочередное получение значения ячеек результата путем сложения значений соответствующих ячеек слагаемых и единицы (в случае переноса разряда с предыдущего шага)

```
1  TVeryLong sum(const TVeryLong& left, const TVeryLong& right) {
2          unsigned long maxof = left.len > right.len ? left.len : right.len;
3          unsigned long minof = left.len > right.len ? right.len : left.len;
4          TVeryLong res;
5          long long pathadd = 0;
6          for (unsigned int i = 0; i < maxof; ++i) {
7              //printf("summing... %lld\n", (left.cells[i] + right.cells[i] + pathadd) %
                   CELLSIZE);
8              if (right.len > left.len) {
9                  res.cells.push_back(i < minof ? (left.cells[i] + right.cells[i] +
                       pathadd) % CELLSIZE : (right.cells[i] + pathadd) % CELLSIZE);
10                 ++res.len;
11
12             }
13             else
14             if (right.len < left.len) {
15                 res.cells.push_back(i < minof ? (left.cells[i] + right.cells[i] +
                       pathadd) % CELLSIZE : (left.cells[i] + pathadd) % CELLSIZE);
16                 ++res.len;
17
18             }
19             else {
20                 //printf("summing sim\n");
21                 res.cells.push_back((left.cells[i] + right.cells[i] + pathadd) %
                       CELLSIZE);
22                 ++res.len;
23             }
24
25             if (i < minof) {
26                 if (left.cells[i] + right.cells[i] + pathadd >= CELLSIZE) {
27                     pathadd = 1;
28                 }
29                 else {
30                     pathadd = 0;
31                 }
32
33                 if (i + 1 == maxof && pathadd > 0) {
34                     res.cells.push_back(pathadd);
35                     ++res.len;
36                 }
```

```
            }
            else
            if (right.len > left.len) {
                if(i >= minof && right.cells[i] + pathadd >= CELLSIZE) {
                    pathadd = 1;
                }
                else {
                    pathadd = 0;
                }
            }
            else
            if (right.len < left.len) {

                if(i >= minof && left.cells[i] + pathadd >= CELLSIZE) {
                    pathadd = 1;
                }
                else {
                    pathadd = 0;
                }
            }
            else{
                pathadd = 0;
            }
        }

        return res;
    }
```

Вычитание выполняется также поразрядного, однако начиная с наибольшего с возвратами в случае необходимости взятия единицы с более старшего разряда. (насколько я помню, когда я только начинал делать вычитание я немного забыл, как эта операция работает и посчитал, что всякий раз придется брать единицу не более одного раза, и потому без разницы, откуда начинать. В итоге так и оставил). В любом случае ситуации, когда при возврате приходится пробегать все число до начала очень редки и происходит это только один раз за всю операцию, так что существенных потерь от такого решения нету.

```
    TVeryLong dif(const TVeryLong& left, const TVeryLong& right) {
        TVeryLong res(left.len);
        if (left.len > right.len || (left.len == right.len && left.cells[left.len - 1]
            >= right.cells[right.len - 1])) {
            for (unsigned int i = left.len; i > right.len; --i) {
                res.cells[i - 1] = left.cells[i - 1];
            }

            for (unsigned int i = right.len; i > 0; --i) {
                if (left.cells[i - 1] >= right.cells[i - 1]) {
                    res.cells[i - 1] = left.cells[i - 1] - right.cells[i - 1];
                }
```

```
12              else {
13                  res.cells[i - 1] = left.cells[i - 1] + CELLSIZE - right.cells[i -
                        1];
14
15                  for(size_t j = i; j < res.len - 1; ++j) {
16                      if(res.cells[j] == 0) {
17                          res.cells[j] = CELLSIZE - 1;
18                      }
19                      else {
20                          res.cells[j]--;
21                          break;
22                      }
23
24                  }
25              }
26          }
27      }
28      else {
29          throw(-1);
30      }
31
32      res.cells.shrink_to_fit();
33      res.len = res.cells.size();
34
35      return res;
36  }
```

Далее операция умножения. В двойном цикле перемножаем ячейки так же, как и при умножении в столбик. Все предельно тривиально.

```
1   TVeryLong comp(const TVeryLong& left, const TVeryLong& right) {
2
3       TVeryLong res(left.len + right.len);
4       //printf("%d %d\n%d\n", left.cells.size(), right.cells.size(), res.cells.size()
            );
5       for (size_t i = 0; i < right.len; ++i) {
6           for (size_t j = 0; j < left.len; ++j) {
7               //printf("%d\n", i + j + 1);
8
9               res.cells[j + i] += left.cells[j] * right.cells[i] % CELLSIZE;
10              if (res.cells[j + i] >= CELLSIZE) {
11                  res.cells[j + i + 1] += res.cells[j + i] / CELLSIZE;
12                  res.cells[j + i] = res.cells[j + i] % CELLSIZE;
13              }
14              res.cells[j + i + 1] += left.cells[j] * right.cells[i] / CELLSIZE;
15
16          }
17
18      }
19
```

```
        res.cells.shrink_to_fit();
        res.len = res.cells.size();

        return res;
    }
```

Возведение в степень. Используем алгоритм возведения в четную степень с циклическим умножением результата на себя и домножением на изначальное число, если на очередном шаге надо произвести возведение в нечетную степень.

```
    my::TVeryLong operator^(const my::TVeryLong& lhs, const my::TVeryLong& rhs)
    {
        if (lhs == my::TVeryLong(std::vector<long long>({ 0 })) && rhs == my::TVeryLong
            (std::vector<long long>({ 0 }))) {
            throw (-1);
        }

        //my::TVeryLong res = my::TVeryLong(std::vector<long long>({ 1 }));
        my::TVeryLong res = my::TVeryLong(std::vector<long long>({ 1 }));
        my::TVeryLong pre = lhs;
        my::TVeryLong deg = rhs;

        while (deg > my::TVeryLong(std::vector<long long>({ 0 }))) {
            /*printf("res\t"); my::print(res); printf("\n");
            printf("pre\t"); my::print(pre); printf("\n");
            printf("deg\t"); my::print(deg); printf("\n");*/
            if (deg.cells[0] % 2 == 1) {
                res = pre * res;
                deg = deg - my::TVeryLong(std::vector<long long>({ 1 }));
            }
            else {
                pre = pre * pre;
                deg = deg / 2;
            }

        }

        return res;
    }
```

Сравнение. Все сравнения выполняются поячеечным сравнением с предварительной проверкой числа ячеек. Например:

```
    bool operator> (const my::TVeryLong& lhs, const my::TVeryLong& rhs) {
        if (lhs.len > rhs.len) {
            return true;
        }
        else {
            if (lhs.len < rhs.len) {
                return false;
```

```
8           }
9           else {
10              for (int i = lhs.len; i > 0; --i) {
11                  if (rhs.cells[i - 1] > lhs.cells[i - 1]) {
12                      return false;
13                  }
14
15                  if (rhs.cells[i - 1] < lhs.cells[i - 1]) {
16                      break;
17                  }
18
19                  if (i == 1 && rhs.cells[i - 1] >= lhs.cells[i - 1]) {
20                      return false;
21                  }
22              }
23              return true;
24          }
25      }
26  }
```

Наиболее сложной же оказалась операция деления. Для удобства я разбил ее на две: деление на число меньшее основания системы счисления и деление на число, большее. Первое выполняется почленным делением, второе же является точным повторением операции деления в столбик.

Для его реализации потребуется написать несколько вспомогательных функций:

1. long long **uniteCells**(long long first, long long second) – Объединение ячеек. Две ячейки числа объединяются в одну, число в результирующей ячейке представляет собой последовательно записанные числа из первой и второй ячейки. Наличие этой операции накладывает ограничение на длину ячеек. Если раньше можно было использовать ячейки с количеством знаков, меньшим максимальной длины long long, то теперь предельная длина ячейки сокращается вдвое.

2. TVeryLong **sTake**(const TVeryLong& wich, const TVeryLong& with) – Отсечение первой делимой части числа. Это как когда мы, деля в столбик отсекаем первые n циферок, таких что число, составленное из них, будет делиться на, собственно, делитель. В моей реализации сначала берутся первые l ячеек (где l - число ячеек в делителе), затем проверяется, больше ли полученное число делителя и, если нет, добавляется еще одна ячейка.

3. long long **findMult**(const TVeryLong& res, const TVeryLong& by) – Поиск множителя. Имеем делитель, имеем минимальный делимый фрагмент делимого (МДФ), полученный функцией отсечения, находим, на что нужно умножить

делитель, чтобы получить МДФ. Используем бинарный поиск. Для начала надо указать хотя бы приблизительные границы. Определим их, поделив первый (наиболее значащий) разряд МДФ на первый разряд делителя. Скорее всего МДФ и делитель имеют одинаковое количество ячеек, однако, если при отсечении мы производили добор, то в МДФ ячеек на одну больше. В таком случае объединим две первые ячейки МДФ с помощью функции uniteCells. Можно производить деление. Полученный результат – верхняя граница искомого множителя. Нижнюю границу получим, поделив на сумму первой ячейки делителя и единицы. Выполняем бинарный поиск, проверяя не нашли ли мы множитель при помощи умножения предполагаемого результата на делитель.

Непосредственно деление. В цикле находим МДФ, находим множитель. В конец результата добавляем ячейку, записываем в нее множитель. Вычитаем из делимого результат умножения МДФ, множителя и 10 в степени отступа от текущей позиции до конца (наименее значащей ячейки) числа. Если при этом делении стерлись еще и нулевые ячейки, то в конец результат добавляется столько нулевых ячеек, сколько мы удалили. Если остаток меньше делителя, то дописываем в конец результата столько нулевых ячеек, сколько их есть в остатке и останавливаем цикл.

```
1  TVeryLong longDiv(const TVeryLong& left, const TVeryLong& right) {
2        long long pos = left.len;
3        TVeryLong del = left;
4        std::vector<long long> preRes;
5
6        if (DEBUG) { printf("del = "); my::print(del); printf("\n"); }
7        size_t step = 0;
8        while(pos > 0) {
9            TVeryLong pr = sTake(del, right);
10
11           if(pr.len - delt > 1 && step > 0) {
12               for(size_t i = 1; i < pr.len - delt; ++i) {
13                   preRes.push_back(0);
14               }
15           }
16
17           if (DEBUG) { printf("\t step %ld. s = ", step); my::print(pr); printf("\n")
                 ; }
18
19           if (DEBUG) { printf("\t pr taken (len %ld). delt: %ld. 0 blocks added: %ld\
                 n",pr.len, delt, pr.len - delt); }
20
21           long long multPR = findMult(pr, right);
22
23           if (DEBUG) {
24               printf("\t dividing... %lld", multPR); printf("\n");
```

```
25              my::print(my::TVeryLong(std::vector<long long>({ multPR })) * right);
                    printf("\n");
26              printf("\t min (%lld %lu) = ", pos, pr.len); my::print(my::TVeryLong(std
                    ::vector<long long>({ multPR })) * right * (my::TVeryLong(std::
                    vector<long long>({ CELLSIZE })) ^ my::TVeryLong(std::vector<long
                    long>({ (long long)(pos - pr.len) }))); printf("\n");
27              printf("ch! > %lld\n", std::vector<long long>({ (long long)(pos - pr.len
                    ) })[0]);
28              printf("ch! > "); my::print((my::TVeryLong(std::vector<long long>({
                    CELLSIZE }))); printf(" !||! "); my::print(my::TVeryLong(std::
                    vector<long long>({ (long long)(pos - pr.len) }))); printf(" !||! ")
                    ; my::print((my::TVeryLong(std::vector<long long>({ CELLSIZE })) ^
                    my::TVeryLong(std::vector<long long>({ (long long)(pos - pr.len) }))
                    )); printf("\n");
29          }
30
31          size_t pastlen = del.len;
32          del = del - (my::TVeryLong(std::vector<long long>({ multPR })) * right * (
                my::TVeryLong(std::vector<long long>({ CELLSIZE })) ^ my::TVeryLong(std
                ::vector<long long>({ (long long)(pos - pr.len) }))));
33          size_t newlen = del.len;
34
35          delt = pr.len - (pastlen - newlen);
36
37          if (DEBUG) {
38              printf("\t new del = "); my::print(del); printf("\n");
39              printf("going to nexty step\n");
40          }
41
42          preRes.push_back (multPR);
43
44
45          if (del < right) {
46              if (DEBUG) {
47                  printf("%lld |%lu %lu\n", pos, left.len, right.len);
48              }
49              TVeryLong finRes = TVeryLong(preRes);
50              finRes = finRes * (my::TVeryLong(std::vector<long long>({ CELLSIZE })) ^
                     my::TVeryLong(std::vector<long long>({ (long long)(pos - pr.len) })
                    ));
51              return finRes;
52          }
53
54          pos = del.len;
55          ++step;
56      }
57      return TVeryLong(preRes);
58  }
```

Для считывания входных данных реализуем функцию configurate, которая будет из

считанной из входного файла строки формировать длинное число:

```
    my::TVeryLong configurate(std::string inp) {
    std::vector<long long> rv;
    long long part = 0;

    int cdn = inp.size() % CELLLEN;
    if (DEBUG) { std::cout << inp << std::endl; }
    if (DEBUG) { printf("%lu|%d\n",inp.size(),cdn); }
    int ii = 0;
    for (size_t i = 0; i < inp.size(); ++i, ++ii) {
        if (DEBUG) { printf("foc st %lu | %c\n", i, inp[i]); }

        part *= 10;
        part += ((long long)inp[i] - '0');

        if ((int)(i + 1) == cdn) {
        // printf("first point reached\n");
            rv.push_back(part);
            ii = -1;
            part = 0;
            cdn = -1;
            continue;
        }

        if ((ii + 1) % CELLLEN == 0) {
          // printf("flush\n");
            rv.push_back(part);
            part = 0;
        }
    }


    my::TVeryLong res(rv);
    return res;
}
```

Само чтение будем выполнять побитово при помощи конечного автомата, построенного на основе заданной в условии конфигурации запросов.

2 Тест производительности

Продолжительность вычислений, очевидно, должна линейно зависеть от количества запросов. На всякий случай проверим это. Для этого сделаем генератор тестового файла, который создает N случайных запросов, где N - аргумент, задаваемый при запуске программы:

```
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 1000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 36ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 2000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 74ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 100000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 3484ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 200000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 6930ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
```

Из результатов теста видно, что зависимость линейная.

Теперь взглянем на зависимость времени выполнения отдельных операций от длины входных чисел. Так как одна операция в любом случае выполняется слишком быстро, чтобы отследить время ее работы будем проводить тесты на основе предыдущей программы генератора. В каждом тесте она будет генерировать N запросов с одной и той же операцией, и порядком входных чисел в $10\hat{4}5$, $10\hat{9}0$, $10\hat{1}80$ и $10\hat{3}60$.

Тестируем деление

```
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 127ms
```

```
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
make test
g++ -o TG.exe test_generator.cpp
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 1000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 367ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
make test
g++ -o TG.exe test_generator.cpp
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 945ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
make test
g++ -o TG.exe test_generator.cpp
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 1000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 2449ms
```

Тестируем умножение

```
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 27ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
make test
g++ -o TG.exe test_generator.cpp
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 1000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 40ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
make test
g++ -o TG.exe test_generator.cpp
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 1000 >fileT1
```

```
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 57ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
make test
g++ -o TG.exe test_generator.cpp
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 1000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 129ms
```

Тестируем сложение и вычитание

```
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
make test
g++ -o TG.exe test_generator.cpp
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 1000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 30ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
make test
g++ -o TG.exe test_generator.cpp
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 1000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 50ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
make test
g++ -o TG.exe test_generator.cpp
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 1000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 47ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
make test
g++ -o TG.exe test_generator.cpp
```

```
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 1000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 69ms
```

Тестируем все четыре основных операции:

```
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
make test
g++ -o TG.exe test_generator.cpp
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 1000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 59ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
make test
g++ -o TG.exe test_generator.cpp
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 1000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 118ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
make test
g++ -o TG.exe test_generator.cpp
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 1000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
test complete| 281ms
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
make test
g++ -o TG.exe test_generator.cpp
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./TG.exe 1000 >fileT1
max@max-Swift:~/Рабочий стол/ДА/lab6/DA_lab_6-main/singleFileVer/bench$
./lab6vsf.exe  <fileT1 >fileR1
```

```
test complete| 606ms
```

Для удобства свеем результаты в таблицу

время работы (мс)				
Мультиплик	/	*	+-	/*+-
5	128	34	30	59
10	367	40	50	118
20	945	50	47	281
40	2449	129	69	606

Мультипликатор - это число, на которое умножается 9 для получения порядка входных чисел (прим. мультипликатор 10 - порядок входных чисел - 90)

Нетрудно заметить, что наиболее медленной из базовых операций является деление. Однако даже оно (по крайней мере на данных порядках) ведет себя линейно, замедляясь примерно 2,5 раза при удвоении длин входных чисел. Сложение и вычитание же выполняются так быстро, что случайные колебания времени выполнения превосходят замедление из-за удлинения операндов. А вот среднее время работы (при одинаковом количестве сложений, вычитаний, умножений и делений) показывает почти идеальную линейную зависимость времени работы от длины входных чисел.

3 Выводы

В ходе выполнения этой лабораторной работы я научился реализовывать арифметические операции над числами в произвольных системах счисления с основанием больше десяти. Очень полезным мне показалось то, что я изначально добавил возможность легко изменять основание системы счисления, в которой производятся операции, что позволило мне легко отладить все функции на числах с двузначными ячейками. Также я смог наглядно увидеть прирост скорости при переходе от линейной сложности к логарифмической (при замене перебора, эффективного на двузначных ячейках, но жутко тормозящего на семизначных, на бинарный поиск), что произвело на меня некоторое впечатление. Наиболее интересной мне показалась задача реализации деления уголком, вероятно потому что все остальные операции реализуются достаточно тривиально.

Список литературы

[1] *Длинная арифметика в C++ — cppstudio.com*
URL: `http://cppstudio.com/post/5036/` (дата обращения: 11.03.2021)

[2] *Длинная арифметика — Викиконспекты.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Арифметика_чисел_в_b-ичной_системе_счисления_(Длинная_арифметика)`
(дата обращения: 14.03.2021)