

## Алгоритмы длинной арифметики и их особенности реализации на C++

*Фатеенков Данила Витальевич*

*Приамурский государственный университет имени Шолом-Алейхема*

*Студент*

### Аннотация

В данной статье рассмотрена реализация алгоритмов выполнения арифметических операций над большими числами. Под большими числами в контексте данной работы подразумеваются числа, которые не входят в диапазоны любых числовых типов данных языка программирования C++. Описана реализация следующих алгоритмов: сложение, вычитание, умножение, деление, сравнение и сортировка (основанная на сравнениях) больших чисел.

**Ключевые слова:** C++, алгоритмизация, длинная арифметика, типы данных, сортировка чисел, арифметические операции

## Algorithms of long arithmetic and their features of implementation in C++

*Fateenkov Danila Vitalievich*

*Sholom-Aleichem Priamursky State University*

*Student*

### Abstract

This paper considers the implementation of algorithms to perform arithmetic operations on large numbers. By large numbers in the context of this paper we mean numbers that are not included in the ranges of any numeric data types of the C++ programming language. The implementation of the following algorithms is described: addition, subtraction, multiplication, division, comparison, and sorting (based on comparisons) of large numbers.

**Keywords:** C++, algorithmic, long arithmetic, data types, number sorting, arithmetic operations

## 1. Введение

### 1.1 Актуальность

Длинная арифметика позволяет программистам работать с числами произвольной длины, что расширяет возможности вычислений. Например, в криптографии длинная арифметика используется для реализации алгоритмов шифрования и подписи, обеспечивая безопасность информации. Алгоритмы длинной арифметики также могут быть полезны в задачах, требующих точных математических вычислений или обработки больших данных.

Многие языки программирования предоставляют библиотеки или встроенную поддержку для длинной арифметики, которая может значительно

упростить работу с большими числами. Знание алгоритмов длинной арифметики позволяет более эффективно использовать эти инструменты.

## 1.2 Обзор исследований

В.А. Попов и Е.А. Канева в своей статье рассмотрели задачи о статистических закономерностях первых цифр натуральных степеней двойки, чисел Фибоначчи и простых чисел [1]. В статье были использованы элементы теории «длинной» арифметики.

А.В. Неласая и М.И. Верещак провели анализ эффективности целочисленных операций в современных библиотеках длинной арифметики [2]. Для проведения анализа использовались библиотеки в различных языках программирования: `apfloat` (Java), `C++ Big Integer Library` (C++), `decNumber` (C) и др.

М.Е. Куляс в своей работе описал варианты использования техники отложенного переноса (ТОП) для ускорения вычисления произведения длинных целых чисел [3]. В рамках статьи использовались алгоритмы длинной арифметики, а именно умножение больших чисел.

М.С. Долинский рассмотрел алгоритмы длинной арифметики в контексте изучения теории чисел и представил реализацию на языке программирования `Pascal` [4].

Н.С. Тимошенко, П.Е. Вдовых и А.С. Кругляков представили программу для вычисления  $N!$  для больших чисел, которая была разработана на языке программирования `C++` [5]. В статье рассмотрены алгоритмы длинной арифметики и представлен код программы.

## 1.3 Цель исследования

Цель – реализовать алгоритмы выполнения арифметических операций над большими числами, которые не входят в диапазоны стандартных типов данных языка программирования `C++`.

## 2. Материалы и методы

Для реализации поставленной задачи используется язык программирования `C++`.

## 3. Результаты и обсуждения

В рамках данной работы будут затронуты только основные арифметические операции, такие как: сложение, вычитание, умножение и деление.

Длинная арифметика в программировании — это методика работы с числами, превышающими по размеру максимальные значения, которые могут быть представлены стандартными типами данных, такими как `int` или даже `long long`.

Таблица 1. Основные типы данных в C++, их размеры в байтах и диапазон значений

Тип данных	Размер	Диапазон
bool	1	true/false
char	1	-128 до 127
unsigned char	1	0 до 255
short	2	-32,768 до 32,767
unsigned short	2	0 до 65,535
int	4	-2,147,483,648 до 2,147,483,647
unsigned int	4	0 до 4,294,967,295
long	4	-2,147,483,648 до 2,147,483,647
long	8	-9,223,372,036,854,775,808 до 9,223,372,036,854,775,807
unsigned long	4	0 до 4,294,967,295
unsigned long	8	0 до 18,446,744,073,709,551,615
long long	8	-9,223,372,036,854,775,808 до 9,223,372,036,854,775,807
unsigned long long	8	0 до 18,446,744,073,709,551,615
float	4	от $-3.4e+38$ до $3.4e+38$
double	8	от $-1.7e+308$ до $1.7e+308$
long double	12 или 16	от $-1.7e+308$ до $1.7e+308$

В таблице 1 видно, что значения в типах данных имеют свои диапазоны, за которые выходить нельзя. Если превысить правую границу диапазона, то число автоматически сместится в левую границу (см. рис. 1).

```

1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      long long num = 9223372036854775807;
8      cout<<num+1;
9  }

```




Рисунок 1. Пример выхода за границы long long

В случае, если необходимо работать с числами, которые больше 9223372036854775807, нужно применять алгоритмы длинной арифметики.

Длинная арифметика позволяет работать с числами произвольной длины, в том числе с числами, состоящими из сотен, тысяч или даже миллионов цифр. Это достигается путем представления чисел в программе в

виде последовательности более маленьких элементов данных, таких как массивы или строки.

Во многих языках в настоящее время реализованы библиотеки для работы с большими числами и являются частью стандартного набора библиотек и модулей. Например, Python поддерживает работу с большими числами без использования дополнительных модулей (см. рис. 2).

```
>>> 54867345987679835476+674586759486798456
55541932747166633932
>>> 54867345987679835476*674586759486798456
37012785131469933455028401216050825056
>>> 54867345987679835476-674586759486798456
54192759228193037020
>>> 54867345987679835476//674586759486798456
81
```

Рисунок 2. Пример работы с большими числами в Python

В C++ нет встроенных библиотек для обработки больших чисел, но есть сторонние, такие как Boost и BigInt. Также начиная со стандарта C++11, в библиотеке «cstdint» появился тип данных `int64_t`. Такой тип данных является 64-битным знаковым целочисленным и может принимать значения в том же диапазоне, что и `long long`.

Соответственно, при попытке выполнить арифметические операции с большими числами напрямую, будет возвращён неправильный ответ (см. рис. 3).

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     long long num1,num2;
8     cin >> num1 >> num2;
9     cout<<num1+num2;
10 }
```

684796837489673486 6734896734986734896739  
-8538575199365102323

Рисунок 3. Попытка сложения больших чисел с использованием стандартных типов данных

Самой базовой арифметической операцией является сложение двух чисел. Алгоритм сложения двух чисел наглядно демонстрируется способом сложения «столбиком»: производится поочерёдное сложение разрядов с правого конца с увеличением последующего разряда на 1, если в текущем сумма разрядов получилась больше 9.

Данный алгоритм довольно прост в реализации на С++ и его можно разбить на следующие этапы:

1. Нахождение разницы длин чисел. Нужно это для того, чтобы меньшее число заполнить слева нулями для выравнивания разрядов и правильного сложения:

```
string t1 = "", t2 = "";
if (a.length() != b.length()) {
    if (a.length() > b.length()) {
        for (int i=0; i<a.length()-b.length(); i++) {
            t1 += "0";
        }
    }
    else {
        for (int i=0; i<b.length()-a.length(); i++) {
            t2 += "0";
        }
    }
}
a = t2 + a;
b = t1 + b;
```

2. Заполнение массива суммами разрядов. Массив нужен как предварительный результат, в котором будут проверяться значения на превышение числа 9:

```
vector<int> temp;
for (int i=a.length()-1; i>=0; i--) {
    temp.push_back(a[i] - '0' + b[i] - '0');
}
```

3. Проверка сумм разрядов и приведение к конечному результату с последующим заполнением строки, которая будет подана на выход функции сложения:

```
for (int i=0; i<temp.size()-1; i++) {
    if (temp[i] > 9) {
        temp[i] -= 10;
        temp[i+1]++;
    }
}
for (int i=temp.size()-1; i>=0; i--) res +=
to_string(temp[i]);
```

Конечный результат будет работать следующим образом (см. рис. 4). Алгоритм позволяет обойти ограничения в виде диапазонов данных, но стоит учитывать, что данные представлены в виде строк и для хранения особенно больших значений может потребоваться дополнительное место в оперативной памяти.

```
54867345987679835476
674586759486798456
Sum of these numbers: 55541932747166633932
```

Рисунок 4. Пример сложения двух больших чисел с помощью вышеописанного алгоритма

Реализованный алгоритм можно использовать в процессе написания умножения больших чисел. Данная операция позволяет наглядно продемонстрировать – насколько большие числа можно обрабатывать с помощью алгоритмов.

Описать умножение двух чисел можно следующим образом: необходимо умножить цифру на каждом разряде одного из чисел на второе число и полученные числа сложить. Опять же – для больших чисел использование даже `long long` не подходит, так как при умножении результат может значительно отличаться от максимально допустимого значения в диапазоне типа данных (см. рис. 2).

Алгоритм умножения чисел также можно разделить на 3 этапа:

1. Выбрать наименьшее число и получить его разряды. Процесс умножения обладает переместительным свойством, то есть от смены мест множителей результат не будет меняться. Брать нужно меньшее число для оптимизации работы алгоритма по памяти:

```
if (a.length() < b.length()) {
    string temp = b;
    b = a;
    a = temp;
}
```

2. Перемножить большее число на разряды меньшего и сохранить результаты в массив. Именно для этого массива необходимо оптимизировать первый этап, потому что если не учесть данный момент, то при слишком больших числах может происходить переполнение памяти, что может привести к сбою работы программы. Также необходимо учитывать, что умножение двух одноразрядных чисел может в результате превысить 9 и нужно перевести десятки в следующий разряд:

```
for (int i=b.length()-1;i>=0;i--) {
    temp_num.clear();
    string num = "";
    int sub_digit = b[i] - '0';
    for (int j=a.length()-1;j>=0;j--) {
        temp_num.push_back(sub_digit * (a[j] - '0'));
    }
    while (true) {
        bool check = true;
        for (int j=temp_num.size()-2;j>=0;j--) {
            if (temp_num[j] > 9) {
```

```

        check = false;
        temp_num[j+1] += temp_num[j] / 10;
        temp_num[j] %= 10;
    }
    }
    if (check) break;
}
for (int j=temp_num.size()-1;j>=0;j--) num +=
to_string(temp_num[j]);
for (int j=0;j<b.length() - i - 1;j++) num += "0";
nums.push_back(num);
}

```

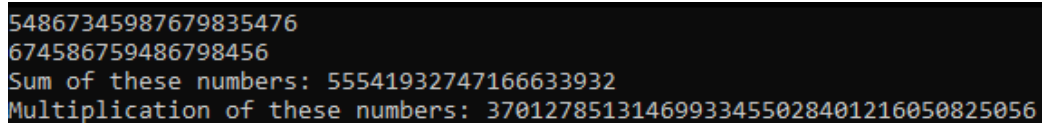
3. Сложить все числа по порядку. Результат сложения и будет являться выходными данными в функции умножения. На этом этапе необходимо использовать ранее написанную функцию сложения больших чисел:

```

if (nums.size() == 1) res = nums[0];
else {
    res = sum(nums[0],nums[1]);
    for (int i=2;i<nums.size();i++) {
        res = sum(res,nums[i]);
    }
}
int i = 0;
while (res[i] == '0') {
    i++;
}
res = res.substr(i);
if (res == "") res = "0";

```

Результатом работы алгоритма является число, которое в сотни и даже тысячи раз может превышать long long (см. рис. 5). Для сравнения можно использовать результаты умножения в Python (см. рис. 2).



```

54867345987679835476
674586759486798456
Sum of these numbers: 55541932747166633932
Multiplication of these numbers: 37012785131469933455028401216050825056

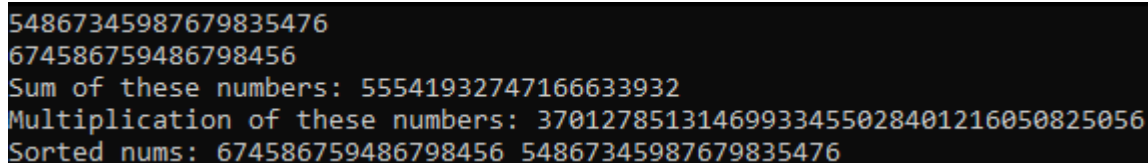
```

Рисунок 5. Результат умножения двух больших чисел с использованием вышеописанного алгоритма

Перед тем как переходить к делению и вычитанию чисел стоит обратить внимание на процесс сортировки больших чисел – он довольно нестандартный, так как необходимо работать со строками, в которых нет буквенных значений. Можно сделать вывод, что цифровая сортировка не подходит для таких строк, так как она опирается не на значения в позициях строки, а на позицию символа в таблице ASCII.

В такой ситуации необходимо написать свой алгоритм сравнения больших чисел, в которых будут сравниваться числа в разрядах, а также длина строк. В результате будет выводиться отсортированная строка (см. рис. 6). Алгоритм был написан для решения задачи деления чисел, поэтому ниже представлен частный случай реализации такого способа сравнения больших чисел, в которых не производится сортировка (но отсортировать можно вне функции, основываясь на результате работы сравнения чисел):

```
if (a.length() != b.length()) {
    if (a.length() > b.length()) return false;
    else return true;
}
else {
    for (int i=0;i<a.length();i++) {
        if ((a[i] - '0') == (b[i] - '0')) continue;
        if ((a[i] - '0') > (b[i] - '0')) {
            return false;
        }
        if ((a[i] - '0') < (b[i] - '0')) {
            return true;
        }
    }
    return true;
}
return false;
```



```
54867345987679835476
674586759486798456
Sum of these numbers: 55541932747166633932
Multiplication of these numbers: 37012785131469933455028401216050825056
Sorted nums: 674586759486798456 54867345987679835476
```

Рисунок 6. Пример работы алгоритма сортировки больших чисел, которые представлены как строки

Теперь можно перейти к вычитанию чисел. Сортировка в данной операции не нужна, но необходимо знать, какое из чисел больше, чтобы знать какой будет ответ – положительный или отрицательный. При этом по модулю всегда результат получается одинаковый. Алгоритм вычитания больших чисел можно представить следующим образом:

1. Нахождение знака результата путём определения того, какое число в заданной паре больше. Для этого можно использовать функцию определения того, какое число больше, которая была описана чуть ранее.

2. Вычисление разности двух чисел. Подход схож с тем, что был при сложении – меняется только знак операции. Также необходимо учитывать, что в разрядах может получиться отрицательное число (тогда к этому числу необходимо прибавить 10 и вычесть из следующего разряда 1). Также если числа разной длины, то необходимо одно из них заполнить недостающими



разрядами (нулями слева), чтобы проводилось вычитание на правильных разрядах:

```

if (a.length() < b.length()) {
    res += "-";
    string temp = a;
    a = b;
    b = temp;
}
string zeros_str = "";
int zeros = a.length() - b.length();
zeros = abs(zeros);
zeros_str.insert(0,zeros,'0');
if (a.length() < b.length()) a = zeros_str + a;
else b = zeros_str + b;
for (int i=0;i<a.length();i++) {
    num.push_back((a[i] - '0') - (b[i] - '0'));
}
for (int i=a.length()-1;i>=1;i--) {
    if (num[i] < 0) {
        num[i] += 10;
        num[i-1] -= 1;
    }
}
}

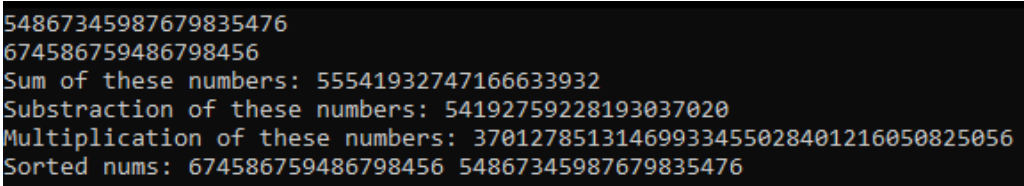
```

3. Проверить результат на наличие лидирующих нулей и удалить их, так как в ответе нулей перед числом быть не может:

```

int i = 0;
if (res[0] == '-') i = 1;
while (res[i] == '0') {
    i++;
}
if (res[0] != '-') {
    res = res.substr(i);
}
else {
    res = res.substr(i);
    res = "-" + res;
}
if (res == "") res = "0";

```



```

54867345987679835476
674586759486798456
Sum of these numbers: 55541932747166633932
Substraction of these numbers: 54192759228193037020
Multiplication of these numbers: 37012785131469933455028401216050825056
Sorted nums: 674586759486798456 54867345987679835476

```

Рисунок 7. Результат работы алгоритма вычитания двух больших чисел

Алгоритм деления позволяет реализовать алгоритмы деления и нахождения остатка от деления. Они самые простые в реализации, потому что

задействуют уже ранее описанные алгоритмы. Деление можно описать следующим образом:

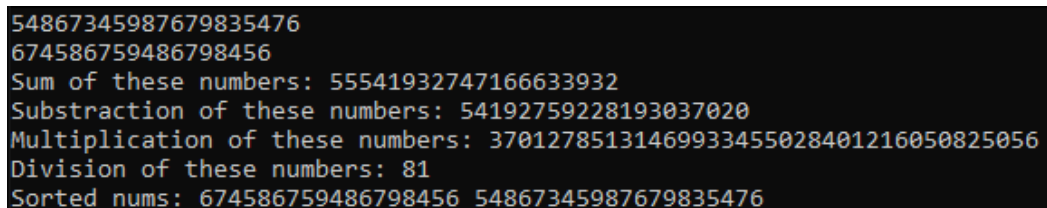
1. Необходимо сначала проверить – превосходит ли делитель делимое число. Если да, то можно вывести 0 и закончить работу функции, иначе перейти к шагу 2.

2. Нужно задать строку, которая будет результатом деления. На входе она равна 1 и к этому числу необходимо прибавлять единицу, пока произведение делителя и этого числа не превзойдёт делимое число:

```
if (b.length() > a.length()) return "0";
string del = "1";
while (!equate_nums(a,mul(b,del))) {
    del = sum(del,"1");
}
```

3. Нужно проверить результат работы алгоритма – если разность делимого и делителя, умноженного на результат работы функции, отрицательная, то результат необходимо уменьшить на 1:

```
string subtraction = sub(a,mul(b,del));
if (subtraction.find("-") != -1) del = sub(del,"1");
res = del;
```



```
54867345987679835476
674586759486798456
Sum of these numbers: 55541932747166633932
Substraction of these numbers: 54192759228193037020
Multiplication of these numbers: 37012785131469933455028401216050825056
Division of these numbers: 81
Sorted nums: 674586759486798456 54867345987679835476
```

Рисунок 8. Результат работы алгоритма деления больших чисел

Алгоритм нахождения остатка от деления двух больших чисел можно считать самым простым: необходимо найти результат деления двух чисел, умножить второе число на этот результат и найти разность между двумя числами:

```
string res = "";
string del = div(a,b);
res = sub(a,mul(b,del));
return res;
```

Как можно видеть из представленного выше кода – используются 3 ранее написанные функции для работы с большими числами (целочисленное деление, вычитание и умножение).

$$54867345987679840000 \bmod 674586759486798500 = 225818469249160580$$

$$\begin{array}{r} 54867345987679840000 \overline{)674586759486798500} \\ \underline{5396694075894388000} \phantom{00} \\ 900405228735959000 \\ \underline{674586759486798500} \\ 225818469249160580 \end{array}$$

```

H:\x \E\u0019pEN\Algorithms\longAriphmetics.exe
54867345987679835476
674586759486798456
Sum of these numbers: 55541932747166633932
Substraction of these numbers: 54192759228193037020
Multiplication of these numbers: 37012785131469933455028401216050825056
Division of these numbers: 81
Remainder of the division of these numbers: 225818469249160540
Sorted nums: 674586759486798456 54867345987679835476

```

Рисунок 9. Результат работы алгоритма нахождения остатка от деления и проверка правильности работы кода

Таким образом, в рамках данной работы были написаны алгоритмы для выполнения арифметических операций над большими числами, которые не входят в диапазоны `long long` или `int64_t`.

Были реализованы следующие алгоритмы: сложение, вычитание, умножение, деление, сравнение, сортировка и нахождения остатка деления двух больших чисел.

### Библиографический список

1. Попов В.А., Канева Е.А. "Длинная" арифметика в исследованиях статистики первых цифр степеней двойки, чисел Фибоначчи и простых чисел // Вестник Сыктывкарского университета. Серия 1: Математика. Механика. Информатика. 2019. № 2 (31). С. 91-107.
2. Неласая А.В., Верещак М.И. Оценка эффективности использования библиотек длинной арифметики в криптографических приложениях // Радиоэлектроника, информатика, управление. 2010. № 2 (23). С. 67-73.
3. Куляс М.Е. Об использовании техники отложенного переноса в арифметике длинных целых чисел // Вестник Московского энергетического института. Вестник МЭИ. 2018. № 6. С. 96-102.
4. Долинский М.С. Элементы теории чисел: длинная арифметика // Информатизация образования. 2016. № 2. С. 12-21.
5. Тимошенко Н.С., Вдовых П.Е., Кругляков А.С. Программа вычисления  $N!$  Для больших чисел // В сборнике: European Research. Сборник статей XV Международной научно-практической конференции: в 2 ч.. 2018. С. 108-110.