МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АДЫГЕЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Инженерно-физический факультет
Кафедра автоматизированных систем обработки информации и
управления

Отчет по практике

Программная визуализация структуры данных
Работа с длинными числами

2 курс, группа 2ИВТ АСОИУ

Выполнила:
_____ Д. А. Дячинская
«___» _____ 2025 г.

Руководитель:
_____ С. В. Теплоухов
«___» _____ 2025 г.

Майкоп, 2025 г.

1. Введение

1.1. Текстовая формулировка задачи (Вариант 1)

Реализовать программу для работы с длинными числами (теми, которые не помещаются в тип long).

1.2. Теория метода

Для описания алгоритмов обычно используется следующее представление целых чисел. Выбирается основание $\beta > 1$. Тогда целое число длины n можно представить в виде:

$$A = a_{n-1} \cdot \beta^{n-1} + \cdots + a_1 \cdot \beta + a_0$$

где

$$0 \leq a_i \leq \beta - 1$$

Умножение Карацубы - метод умножения относящийся к парадигме «разделяй и властвуй». Вычислительная сложность алгоритма:

$$M(n) = O(n^{\log_2 3}), \log_2 3 = 1,5849\ldots$$

Данный алгоритм представляет собой простую реализацию идеи разделения входных данных, которая стала базисной для нижеперечисленных алгоритмов. Идея заключается в разделении одной операции умножения над n-значными числами на три операции умножения над числами длины

$$\frac{n}{2} + O(n)$$

Для перемножения двух чисел, превышающих длину машинного слова, алгоритм Карацубы вызывается рекурсивно до тех пор, пока эти числа не станут достаточно маленькими, чтобы их можно было перемножить непосредственно. Пример такого алгоритма:
input: $A = \sum_{i=0}^{n-1} a_i \beta^i, B = \sum_{j=0}^{n-1} b_j \beta^j$
output:$C = AB := \sum_{k=0}^{2n-1} c_k \beta^k$
$k \leftarrow [\frac{n}{2}]$
$(\mathrm{A}_0, B_0) := (A, B) \left|\beta\right|^k, (A_1, (A_0, B_0) := (A, B) mod \beta^k, (A_1, B_1) := (A, B) \div \beta^k \mathrm{B}_1) := (A, B) \div \beta^k$
$s_A \leftarrow sign(A_0 - A_1), s_B \leftarrow sign(B_0 - B_1),$
$C_0 \leftarrow KaratsubaMultiply(A_0, B_0)$
$C_1 \leftarrow KaratsubaMultiply(A_1, B_1)$
$C_2 \leftarrow KaratsubaMultiply(|A_0 - A_1|, |B_0 - B_1|)$
return $C := C_0 + (C_0 + C_1 - s_A s_B C_2)\beta_k + C_1 \beta^{2k}$

2. Ход работы

2.1. Выбор средств для разработки

Для реализации программы для работы с длинными числами я выбрала стек технологий, состоящий из:

- **C++** — основной язык разработки, обеспечивающий высокую производительность и возможность низкоуровнего контроля над памятью;
- **Страндартные библиотели C++** `STL`:
  1) `<iostream>` - ввод и вывод через стандартные потоки;
  2) `<string>` - хранение и обработка чисел в виде строки;
  3) `<algorithm>` - алгоритмы общего назначения;
- **Алгоритм умножения Карацубы** — для эффективного и быстрого умножения длинных чисел;
- **Реализация базовых арифметических операций**(сложение,вычитание,деление)на строках,что позволяет выполнять вычисления с произвольной длиной чисел,выходящих за приделы стандартных типов данных.

2.2. Код приложения

В основе приложения лежит реализация программы для работы с длинными числами. Ниже приведён код работы с длинными числами, содержащий алгоритм умножения Карацубы и алгоритм деления:

2.3. Код приложения

```
}
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

// удаляет ведущие нули из строки
string stripLeadingZeros(string num) {
    while (num.length() > 1 && num[0] == '0')
        num.erase(0, 1);
    return num;
}

// сравнение двух чисел в виде строки
int compare(const string& a, const string& b) {
```

```
    string A = stripLeadingZeros(a);
    string B = stripLeadingZeros(b);
    if (A.size() != B.size()) return A.size() < B.size() ? -1 : 1;
    return A.compare(B);
}

// сложение двух больших чисел, представленных строками
string add(const string& num1, const string& num2) {
    string result;
    int carry = 0;
    int i = num1.size() - 1, j = num2.size() - 1;

    while (i >= 0  j >= 0  carry) {
        int digitSum = carry;
        if (i >= 0) digitSum += num1[i--] - '0';
        if (j >= 0) digitSum += num2[j--] - '0';
        result += digitSum % 10 + '0';
        carry = digitSum / 10;
    }

    reverse(result.begin(), result.end());
    return result;
}

// (num1 >= num2)
string subtract(const string& num1, const string& num2) {
    string result;
    int borrow = 0;
    int i = num1.size() - 1, j = num2.size() - 1;

    while (i >= 0) {
        int diff = (num1[i] - '0') - borrow;
        if (j >= 0) diff -= (num2[j--] - '0');

        if (diff < 0) {
            diff += 10;
            borrow = 1;
        }
        else {
            borrow = 0;
        }

        result += diff + '0';
        i--;
```

```
    }

    reverse(result.begin(), result.end());
    return stripLeadingZeros(result);
}

// умножение большого числа на 10^n
string shift(const string& num, int n) {
    return num + string(n, '0');
}

// умножение по Карацубе
string karatsuba(const string& x, const string& y) {
    string a = stripLeadingZeros(x);
    string b = stripLeadingZeros(y);

    if (a.size() == 0 || b.size() == 0) return "0";
    if (a.size() == 1 && b.size() == 1)
        return to_string((a[0] - '0') * (b[0] - '0'));

    int n = max(a.size(), b.size());
    int half = n / 2;

    while (a.size() < n) a = '0' + a;
    while (b.size() < n) b = '0' + b;

    string a1 = a.substr(0, n - half);
    string a0 = a.substr(n - half);
    string b1 = b.substr(0, n - half);
    string b0 = b.substr(n - half);

    string z2 = karatsuba(a1, b1);
    string z0 = karatsuba(a0, b0);
    string z1 = karatsuba(add(a1, a0), add(b1, b0));
    z1 = subtract(subtract(z1, z2), z0);

    return stripLeadingZeros(add(add(shift(z2, 2 * half), shift(z1, half)), z0));
}

// деление длинных чисел: целочисленное деление
string divide(const string& dividend, const string& divisor) {
    if (stripLeadingZeros(divisor) == "0") {
        throw invalid_argument("Division by zero");
    }
```

```
    string result;
    string current;

    for (size_t i = 0; i < dividend.size(); ++i) {
        current += dividend[i];
        current = stripLeadingZeros(current);

        int quotientDigit = 0;
        while (compare(current, divisor) >= 0) {
            current = subtract(current, divisor);
            ++quotientDigit;
        }
        result += quotientDigit + '0';
    }

    return stripLeadingZeros(result);
}

int main() {
    setlocale(LC_ALL, "");
    string num1, num2;
    cout << "Введите первое длинное число: ";
    cin >> num1;
    cout << "Введите второе длинное число: ";
    cin >> num2;

    cout << "Умножение Карацубы: " << karatsuba(num1, num2) << endl;
    try {
        cout << "Результат алгоритма деления: " << divide(num1, num2) << endl;
    }
    catch (const exception& e) {
        cout << "Ошибка: " << e.what() << endl;
    }

    return 0;
}
```

2.4. Основные логические компоненты кода

- `main()` — точка ввода,организует ввод/вывод и вызов функций.
- `stripLeadingZeros()` — удаление ведущих нулей в начале строки.
- `compare()` — сравнение двух больших чисел.

- `add()` — сложение двух больших чисел.
- `subtract()` — вычисление одного большого числа из другого.
- `shift()` — умножение числа на 10^n,добавление нулей в конец строки.
- `karatsuba()` — умножение двух длинных чисел по алгоритму Карацубы.
- `divide()` — деление двух длинных чисел.

3. Скриншот результата

На следующем изображении представлен результат работы кода:

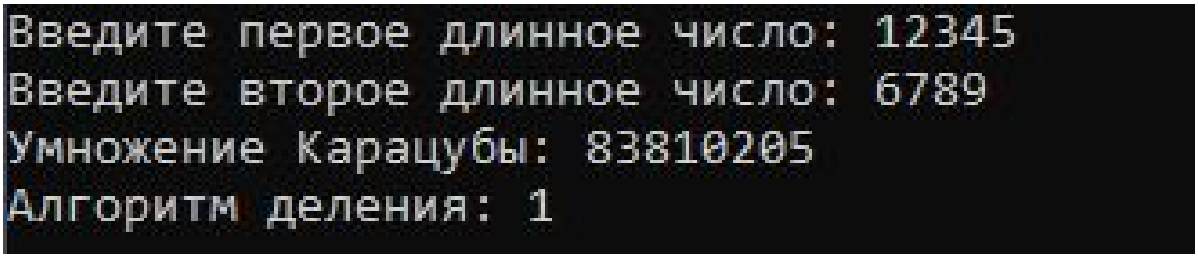

Рис. 1. результат выполниения программы

4. Источники

Список литературы

[1] Кнут Д.Э. Всё про TEX. — Москва: Изд. Вильямс, 2003 г. 550 с.

[2] Львовский С.М. Набор и верстка в системе LATEX. — 3-е издание, исправленное и дополненное, 2003 г.

[3] Shustrova A. Умножение больших чисел: от школьного метода до Карацубы [Электронный ресурс] .— https://habr.com/ru/articles/451860/.

[4] GeeksforGeeks: a computer science portal for geeks [Электронный ресурс]. — https://www.geeksforgeeks.org, 2024 г.

5. Доступ к исходному коду

Репозиторий: `https://github.com/dariadya/long-num`