

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
высшего образования «Санкт-Петербургский политехнический университет  
Петра Великого»

Институт компьютерных наук и кибербезопасности  
Направление: 02.03.01 Математика и компьютерные науки

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1  
по дисциплине Дискретная математика

**Калькулятор «большой» конечной арифметики**  
Вариант 1

Студент,  
группы 5130201/40003

\_\_\_\_\_ Адиатуллин Т.Р

Доцент

\_\_\_\_\_ Востров А.В

«\_\_\_\_\_» \_\_\_\_\_ 20 \_\_ г.

Санкт-Петербург, 2025

# Содержание

<b>Введение</b>	<b>4</b>
<b>1 Математическое описание</b>	<b>5</b>
1.1 «Малая» и «большая» конечные арифметики	5
1.2 Таблицы операций малой арифметики	6
<b>2 Особенности реализации</b>	<b>7</b>
2.1 Структура проекта	7
2.2 Структуры данных	7
2.2.1 Класс SmallArithmetic	7
2.2.2 Класс BigArithmeticCalc	8
2.3 Класс SmallArithmetic	8
2.3.1 Назначение и структура данных	8
2.3.2 Метод nextSymbol(const string& current)	10
2.3.3 Метод compareSymbols(const string& a, const string& b)	11
2.3.4 Метод addByHasse(const string& a, const string& b)	11
2.3.5 Метод multiplyByHasse(const string& a, const string& b)	13
2.3.6 Метод subtractByHasse(const string& a, const string& b)	14
2.3.7 Метод buildAdditionTableWithCarry()	15
2.4 Класс BigArithmeticCalc	16
2.4.1 Назначение и структура данных	16
2.4.2 Метод isNegative(const string& num)	17
2.4.3 Метод removeSign(const string& num)	17
2.4.4 Метод addSign(const string& num, bool negative)	18
2.4.5 Метод isValidNumber(const string& num)	18
2.4.6 Метод isOverflow(const string& num)	19
2.4.7 Метод deleteTrashZeros(const string& num)	20
2.4.8 Метод compareBigUnsigned(const string& a, const string& b)	21
2.4.9 Метод addBigUnsigned(const string& a, const string& b)	22
2.4.10 Метод subtractBigUnsigned(const string& a, const string& b)	22
2.4.11 Метод multiplyBigUnsigned(const string& a, const string& b)	24
2.4.12 Метод multiplyByDigit(const string& num, const string& digit)	24
2.4.13 Метод divideBigUnsigned(const string& a, const string& b)	26
2.4.14 Метод add(const string& a, const string& b)	27
2.4.15 Метод subtract(const string& a, const string& b)	28
2.4.16 Метод multiply(const string& a, const string& b)	28
2.4.17 Метод divide(const string& a, const string& b)	29
<b>3 Результаты работы программы</b>	<b>31</b>
3.1 Запуск программы	31
3.2 Сценарий 1: Просмотр справки	32

3.2.1	Команда help . . . . .	32
3.2.2	Команда info . . . . .	32
3.3	Сценарий 2: Просмотр диаграммы Хассе . . . . .	33
3.3.1	Команда hasse . . . . .	33
3.4	Сценарий 3: Операции с большими числами . . . . .	33
3.4.1	Сложение положительных чисел . . . . .	33
3.4.2	Сложение с переносами . . . . .	33
3.4.3	Сложение с отрицательными числами . . . . .	34
3.4.4	Вычитание положительных чисел . . . . .	34
3.4.5	Вычитание с заимствованием . . . . .	34
3.4.6	Умножение положительных чисел . . . . .	35
3.4.7	Умножение с отрицательными числами . . . . .	35
3.4.8	Деление положительных чисел . . . . .	36
3.4.9	Деление с отрицательными числами . . . . .	36
3.5	Сценарий 4: Обработка ошибок . . . . .	37
3.5.1	Некорректный формат числа . . . . .	37
3.5.2	Деление на ноль . . . . .	37
3.5.3	Переполнение разрядов . . . . .	37
3.5.4	Неизвестная команда . . . . .	37
3.6	Выход из программы . . . . .	38

<b>Заключение</b>	<b>39</b>
-------------------	-----------

<b>Список литературы</b>	<b>41</b>
--------------------------	-----------

# Введение

Курсовая работа посвящена разработке калькулятора большой конечной арифметики  $\langle Z_8; +, * \rangle$  для четырёх арифметических операций: сложения, вычитания, умножения и деления. Калькулятор построен на основе малой конечной арифметики, в которой задано правило ”+1” и выполняются свойства коммутативности сложения и умножения, ассоциативности этих операций, дистрибутивности умножения относительно сложения. В системе заданы аддитивная единица «а» и мультипликативная единица «b», а также выполняется свойство  $x \times a = a$  для любого элемента  $x$ .

Правило ”+1” определяет переход от текущего символа к следующему и задано в соответствии с таблицей 1.

Таблица 1: Правило ”+1”

$x$	a	b	c	d	e	f	g	h
$x + 1$	b	c	e	g	d	h	f	a

# 1. Математическое описание

## 1.1. «Малая» и «большая» конечные арифметики

Множество  $Z$  вместе с набором операций  $\Sigma = \{\varphi_1, \dots, \varphi_m\}$ ,  $\varphi_i : Z^{n_i} \rightarrow Z$ , где  $n_i$  — аргументность операции  $\varphi_i$ , называется алгебраической структурой, универсальной алгеброй или просто алгеброй.

Коммутативное кольцо с единицей — алгебраическая структура  $\langle Z; +, * \rangle$ , в которой выполняются следующие аксиомы:

1. Ассоциативность сложения:  $(a + b) + c = a + (b + c)$
2. Существование нулевого элемента:  $\exists 0 \in Z (\forall a \in Z : a + 0 = 0 + a = a)$
3. Существование противоположного элемента:  $\forall a \in Z \exists (-a) \in Z : a + (-a) = 0$
4. Коммутативность сложения:  $a + b = b + a$
5. Ассоциативность умножения:  $(a * b) * c = a * (b * c)$
6. Дистрибутивность:  $a * (b + c) = a * b + a * c$
7. Коммутативность умножения:  $a * b = b * a$
8. Существование единичного элемента:  $\exists 1 \in Z (\forall a \in Z : a * 1 = 1 * a = a)$

«Малая» конечная арифметика — конечное коммутативное кольцо с единицей  $\langle Z_8; +, * \rangle$ , на котором определены действия вычитания и деления, причём деление определено частично (только для элементов, имеющих мультипликативную инверсию).

В данной работе используется кольцо  $Z_8 = \{a, b, c, d, e, f, g, h\}$  размера  $i = 8$ , где:

- Нулевой элемент (нейтральный по сложению):  $a$
- Единичный элемент (нейтральный по умножению):  $b$
- Отношение порядка (правило инкремента «+1»):  
 $a \rightarrow b \rightarrow c \rightarrow e \rightarrow d \rightarrow g \rightarrow f \rightarrow h$

«Большая» конечная арифметика — конечное коммутативное кольцо с единицей  $\langle Z_i^n; +, * \rangle$ , элементами которого являются слова длины до  $n$  над алфавитом  $Z_i$ . Операции определены позиционно с учётом переноса разрядов. Деление определено с остатком.

В данной работе  $n = 8$  (максимум 8 разрядов), поэтому итоговая структура имеет вид  $\langle Z_8^8; +, * \rangle$ .

Каждому символу сопоставлен индекс (позиционное значение):

	<i>a</i>	<i>b</i>	<i>c</i>	<i>e</i>	<i>d</i>	<i>g</i>	<i>f</i>	<i>h</i>
	0	1	2	3	4	5	6	7

## 1.2. Таблицы операций малой арифметики

На рисунках 1 и 2 представлены таблицы операций и переносов для сложения и умножения соответственно

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>a</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>b</i>	<i>b</i>	<i>c</i>	<i>e</i>	<i>g</i>	<i>d</i>	<i>h</i>	<i>f</i>	<i>a</i>
<i>c</i>	<i>c</i>	<i>e</i>	<i>d</i>	<i>f</i>	<i>g</i>	<i>a</i>	<i>h</i>	<i>b</i>
<i>d</i>	<i>d</i>	<i>g</i>	<i>f</i>	<i>a</i>	<i>h</i>	<i>c</i>	<i>b</i>	<i>e</i>
<i>e</i>	<i>e</i>	<i>d</i>	<i>g</i>	<i>h</i>	<i>f</i>	<i>b</i>	<i>a</i>	<i>c</i>
<i>f</i>	<i>f</i>	<i>h</i>	<i>a</i>	<i>c</i>	<i>b</i>	<i>d</i>	<i>e</i>	<i>g</i>
<i>g</i>	<i>g</i>	<i>f</i>	<i>h</i>	<i>b</i>	<i>a</i>	<i>e</i>	<i>c</i>	<i>d</i>
<i>h</i>	<i>h</i>	<i>a</i>	<i>b</i>	<i>e</i>	<i>c</i>	<i>g</i>	<i>d</i>	<i>f</i>

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>
<i>b</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>
<i>c</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>
<i>d</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>
<i>e</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>
<i>f</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>
<i>g</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>
<i>h</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>	<i>b</i>

Рис. 1: Таблица сложения и таблица переносов для малой арифметики

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>
<i>b</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>c</i>	<i>a</i>	<i>c</i>	<i>d</i>	<i>a</i>	<i>f</i>	<i>d</i>	<i>c</i>	<i>f</i>
<i>d</i>	<i>a</i>	<i>d</i>	<i>a</i>	<i>a</i>	<i>d</i>	<i>a</i>	<i>d</i>	<i>d</i>
<i>e</i>	<i>a</i>	<i>e</i>	<i>f</i>	<i>d</i>	<i>b</i>	<i>c</i>	<i>h</i>	<i>g</i>
<i>f</i>	<i>a</i>	<i>f</i>	<i>d</i>	<i>a</i>	<i>c</i>	<i>d</i>	<i>f</i>	<i>c</i>
<i>g</i>	<i>a</i>	<i>g</i>	<i>c</i>	<i>d</i>	<i>h</i>	<i>f</i>	<i>b</i>	<i>e</i>
<i>h</i>	<i>a</i>	<i>h</i>	<i>f</i>	<i>d</i>	<i>g</i>	<i>c</i>	<i>e</i>	<i>b</i>

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>
<i>b</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>a</i>
<i>c</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>b</i>
<i>d</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>b</i>	<i>e</i>	<i>c</i>	<i>e</i>
<i>e</i>	<i>a</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>b</i>	<i>c</i>	<i>b</i>	<i>c</i>
<i>f</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>e</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>g</i>
<i>g</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>b</i>	<i>e</i>	<i>e</i>	<i>d</i>
<i>h</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>e</i>	<i>c</i>	<i>g</i>	<i>d</i>	<i>f</i>

Рис. 2: Таблица умножения и таблица переносов для малой арифметики

## 2. Особенности реализации

### 2.1. Структура проекта

Проект имеет следующую структуру:

- `SmallArithmetic.hpp` и `SmallArithmetic.cpp`: содержат реализацию малой конечной арифметики на основе диаграммы Хассе. Класс выполняет операции над однозначными элементами алфавита.
- `BigArithmeticCalc.hpp` и `BigArithmeticCalc.cpp`: содержат реализацию большой конечной арифметики. Класс выполняет операции столбиком над многозначными числами.
- `UI.hpp` и `UI.cpp`: реализуют пользовательский интерфейс, обработку команд и арифметических выражений.
- `config.hpp`: содержит конфигурацию системы — мощность алфавита, правило "+1" и сам алфавит.
- `main.cpp`: точка входа в программу.
- `CMakeLists.txt`: автоматизация сборки проекта.

### 2.2. Структуры данных

Для реализации конечной арифметики используются два класса: `SmallArithmetic` и `BigArithmeticCalc`.

#### 2.2.1. Класс `SmallArithmetic`

Класс хранит следующие основные структуры:

- `map<string, string> plusOneRule`: правило перехода к следующему элементу алфавита.
- `vector<string> alphabet`: набор всех элементов алфавита.
- `map<string, string> inverseMap`: таблица обратных элементов для деления.
- `map<tuple<... >, pair<...>> additionTableWithCarry`: таблица сложения с переносом для операций столбиком.
- `vector<vector<string>> addTable, mulTable, subTable, divTable`: таблицы операций малой арифметики.
- `int N`: мощность алфавита.

- `static const int MAX_DIGITS = 8`: максимальное количество разрядов в числах.

### 2.2.2. Класс `BigArithmeticCalc`

Класс содержит:

- `SmallArithmetic small`: объект малой арифметики для выполнения посимвольных операций.

## 2.3. Класс `SmallArithmetic`

### 2.3.1. Назначение и структура данных

Класс `SmallArithmetic` реализует малую конечную арифметику на заданном алфавите с правилом перехода к следующему элементу (правило "+1"). Все операции выполняются через построение диаграммы Хассе.

```

1 class SmallArithmetic {
2 private:
3     int N;
4     map<string, string> plusOneRule;
5     vector<string> alphabet;
6
7     map<string, string> inverseMap;
8
9     map<tuple<string, string, string>, pair<string, string>>
10        additionTableWithCarry;
11     map<string, string> negationMap;
12
13     vector<vector<string>> addTable;
14     vector<vector<string>> mulTable;
15     vector<vector<string>> subTable;
16     vector<vector<string>> divTable;
17
18     string additiveIdentity;
19     string multiplicativeIdentity;
20     string universum;
21     string emptySet;
22
23     static const int MAX_DIGITS = 8;
24
25     string addByHasse(const string& a, const string& b) const;
26     string multiplyByHasse(const string& a, const string& b) const;
27     string subtractByHasse(const string& a, const string& b) const;
28     string divideByHasse(const string& a, const string& b) const;
29
30     optional<string> findMultiplicativeInverse(const string& x);
31     void buildInverseMap();
32     void buildAddTable();

```



```

33 void buildMulTable();
34 void buildSubTable();
35 void buildDivTable();
36
37 void buildAdditionTableWithCarry();
38 bool checkCarry(const string& start, const string& steps)
    const;
39 void buildNegationMap();
40
41 string nextSymbol(const string& current) const;
42 int compareSymbols(const string& a, const string& b) const;
43
44 void printTable(const vector<vector<string>>& table) const;
45
46 public:
47     SmallArithmetic(int n,
48                     const map<string, string>& rule,
49                     const vector<string>& alph,
50                     const string& addId = "a",
51                     const string& mulId = "b");
52
53     const vector<string>& getAlphabet() const;
54     int getN() const;
55
56     const string& getAdditiveIdentity() const;
57     const string& getMultiplicativeIdentity() const;
58     const string& getUniversum() const;
59     const string& getEmptySet() const;
60     int getMaxDigits() const;
61     string getMinNumber() const;
62     string getMaxNumber() const;
63
64     bool isValidElement(const string& elem) const;
65     int compareElems(const string& a, const string& b) const;
66     string nextElem(const string& current) const;
67     pair<string, string> addWithCarry(const string& c1,
68                                     const string& c2,
69                                     const string& carry_in)
70                                     const;
71     string smallSubtract(const string& a, const string& b) const;
72
73     void printAddTable() const;
74     void printMulTable() const;
75     void printSubTable() const;
76     void printDivTable() const;
77     void printAllTables() const;
78     void printHasseDiagram() const;
79 };

```

Listing 1: Интерфейс класса SmallArithmetic

## Ключевые методы

### 2.3.2. Метод nextSymbol(const string& current)

**Назначение:** Возвращает следующий элемент алфавита согласно правилу "+1". Используется для обхода диаграммы Хассе при выполнении операций сложения и умножения.

**Вход:**

- const string& current: Текущий символ алфавита.

**Выход:**

- string: Следующий символ по правилу "+1".

```
1 string SmallArithmetic::nextSymbol(const string& current) const {  
2     auto it = plusOneRule.find(current);  
3     if (it == plusOneRule.end()) {  
4         throw runtime_error("invalid symbol: " + current);  
5     }  
6     return it->second;  
7 }
```

Listing 2: Реализация nextSymbol()

### 2.3.3. Метод `compareSymbols(const string& a, const string& b)`

**Назначение:** Сравнивает два элемента алфавита, определяя их относительный порядок в диаграмме Хассе. Используется для операций сравнения и вычитания.

**Вход:**

- `const string& a`: Первый элемент для сравнения.
- `const string& b`: Второй элемент для сравнения.

**Выход:**

- `int`: 0 (если  $a = b$ ),  $-1$  (если  $a < b$ ), 1 (если  $a > b$ ).

```
1 int SmallArithmetic::compareSymbols(const string& a, const string&
  b) const {
2     if (a == b) return 0;
3
4     string current = additiveIdentity;
5     while (true) {
6         if (current == a) return -1;
7         if (current == b) return 1;
8         current = nextSymbol(current);
9         if (current == additiveIdentity) break;
10    }
11    return 0;
12 }
```

Listing 3: Реализация `compareSymbols()`

Алгоритм последовательно проходит по циклу правила "+1", начиная с нейтрального элемента, и определяет, какой из элементов встречается раньше.

### 2.3.4. Метод `addByHasse(const string& a, const string& b)`

**Назначение:** Выполняет сложение двух элементов алфавита путём последовательного применения правила "+1". Результат вычисляется как  $a + b = \underbrace{(a + 1) + 1 \dots + 1}_b$ .

**Вход:**

- `const string& a`: Первое слагаемое.
- `const string& b`: Второе слагаемое.

**Выход:**

- `string`: Результат сложения  $a + b$ .

```

1 string SmallArithmetic::addByHasse(const string& a, const string&
  b) const {
2     if (b == additiveIdentity) return a;
3
4     string counter = additiveIdentity;
5     string result = a;
6
7     while (counter != b) {
8         result = nextSymbol(result);
9         counter = nextSymbol(counter);
10    }
11
12    return result;
13 }

```

Listing 4: Реализация addByHasse()

Алгоритм работает следующим образом:

1. Если второе слагаемое является нейтральным элементом, возвращается первое слагаемое;
2. Инициализируется счётчик нейтральным элементом;
3. В цикле счётчик увеличивается до значения второго слагаемого;
4. Одновременно результат увеличивается на ту же величину;
5. Возвращается итоговое значение.

### 2.3.5. Метод `multiplyByHasse(const string& a, const string& b)`

**Назначение:** Выполняет умножение двух элементов алфавита через повторное сложение. Результат вычисляется как  $a \times b = \underbrace{a + a + \dots + a}_b$ .

**Вход:**

- `const string& a`: Множитель.
- `const string& b`: Множитель.

**Выход:**

- `string`: Результат умножения  $a \times b$ .

```
1 string SmallArithmetic::multiplyByHasse(const string& a, const
  string& b) const {
2     if (a == additiveIdentity && b == additiveIdentity) {
3         return universum;
4     }
5
6     if (a == additiveIdentity || b == additiveIdentity) {
7         return additiveIdentity;
8     }
9
10    string counter = additiveIdentity;
11    string result = additiveIdentity;
12
13    while (counter != b) {
14        result = addByHasse(result, a);
15        counter = nextSymbol(counter);
16    }
17
18    return result;
19 }
```

Listing 5: Реализация `multiplyByHasse()`

Алгоритм:

1. Обработка краевого случая: произведение двух нейтральных элементов по сложению даёт универсум;
2. Если хотя бы один множитель — нейтральный элемент, результат — нейтральный элемент;
3. Иначе выполняется повторное сложение множимого с самим собой  $b$  раз.

### 2.3.6. Метод `subtractByHasse(const string& a, const string& b)`

**Назначение:** Выполняет вычитание путём поиска такого элемента  $c$ , что  $b + c = a$ . Реализует обратную операцию к сложению.

**Вход:**

- `const string& a`: Уменьшаемое.
- `const string& b`: Вычитаемое.

**Выход:**

- `string`: Результат вычитания  $a - b$  или строка ошибки.

```
1 string SmallArithmetic::subtractByHasse(const string& a, const
  string& b) const {
2     for (const auto& candidate : alphabet) {
3         if (addByHasse(b, candidate) == a) {
4             return candidate;
5         }
6     }
7     return "ERR: with subtract";
8 }
```

Listing 6: Реализация `subtractByHasse()`

Алгоритм перебирает все элементы алфавита и проверяет, какой из них при сложении с вычитаемым даёт уменьшаемое.

### 2.3.7. Метод buildAdditionTableWithCarry()

**Назначение:** Строит таблицу для сложения с учётом входящего переноса. Таблица используется в большой арифметике при сложении столбиком. Для каждой тройки  $(a, b, c_{in})$  вычисляется пара  $(sum, c_{out})$ .

**Вход:**

- `vector<string> alphabet`

**Выход:**

- `map<tuple<string, string, string>, pair<string, string>> additionTableWithCarry` - таблица для сложения с учётом входящего переноса.

```
1 void SmallArithmetic::buildAdditionTableWithCarry() {
2     additionTableWithCarry.clear();
3
4     for (const auto& c1 : alphabet) {
5         for (const auto& c2 : alphabet) {
6             for (const auto& carry_in : alphabet) {
7                 string temp = addByHasse(c1, c2);
8                 string sum = addByHasse(temp, carry_in);
9
10                string carry_out = additiveIdentity;
11                string result = sum;
12
13                if (checkCarry(c1, c2)) {
14                    carry_out = nextSymbol(additiveIdentity);
15                }
16
17                additionTableWithCarry[make_tuple(c1, c2,
18                    carry_in)] =
19                    make_pair(result, carry_out);
20            }
21        }
22    }
```

Listing 7: Реализация buildAdditionTableWithCarry()

Таблица содержит тройки  $(a, b, c_{in})$  и соответствующие им пары  $(sum, c_{out})$ , где:

- $a, b$  — слагаемые разрядов;
- $c_{in}$  — входящий перенос из младшего разряда;
- $sum$  — результат сложения в текущем разряде;
- $c_{out}$  — исходящий перенос в старший разряд.

## 2.4. Класс BigArithmeticCalc

### 2.4.1. Назначение и структура данных

Класс BigArithmeticCalc реализует большую конечную арифметику — операции над многозначными числами. Внутри использует класс SmallArithmetic для посимвольных операций.

```
1 class BigArithmeticCalc {
2 private:
3     SmallArithmetic small;
4
5     bool isNegative(const string& num) const;
6     string removeSign(const string& num) const;
7     string addSign(const string& num, bool negative) const;
8
9     bool isValidNumber(const string& num) const;
10
11     string deleteTrashZeros(const string& num) const;
12
13     bool isOverflow(const string& num) const;
14
15     string addBigUnsigned(const string& a, const string& b) const;
16     string subtractBigUnsigned(const string& a, const string& b)
17         const;
18     string multiplyBigUnsigned(const string& a, const string& b)
19         const;
20     pair<string, string> divideBigUnsigned(const string& a, const
21         string& b) const;
22
23     string multiplyByDigit(const string& num, const string& digit)
24         const;
25
26     int compareBigUnsigned(const string& a, const string& b) const;
27
28 public:
29     BigArithmeticCalc(int n,
30         const map<string, string>& rule,
31         const vector<string>& alph,
32         const string& addId = "a",
33         const string& mulId = "b");
34
35     const vector<string>& getAlphabet() const;
36     string getMinNumber() const;
37     string getMaxNumber() const;
38
39     string add(const string& a, const string& b) const;
40     string multiply(const string& a, const string& b) const;
41     string subtract(const string& a, const string& b) const;
42     string divide(const string& a, const string& b) const;
43
44     void printAddTable() const;
45     void printMulTable() const;
```



```

42 void printSubTable() const;
43 void printDivTable() const;
44 void printAllTables() const;
45 void printInfo() const;
46 void printHasseDiagram() const;
47 void printHelp() const;
48 };

```

Listing 8: Интерфейс класса BigArithmeticCalc

Класс хранит только один объект малой арифметики, через который выполняются все операции с разрядами.

## Вспомогательные методы

### 2.4.2. Метод isNegative(const string& num)

**Назначение:** Проверяет, является ли число отрицательным (начинается ли строка с символа минус).

**Вход:**

- const string& num: Число для проверки.

**Выход:**

- bool: true, если число отрицательное, иначе false.

```

1 bool BigArithmeticCalc::isNegative(const string& num) const {
2     return ! num.empty() && num[0] == '-';
3 }

```

Listing 9: Реализация isNegative()

### 2.4.3. Метод removeSign(const string& num)

**Назначение:** Удаляет знак минус из начала строки, возвращая абсолютное значение числа.

**Вход:**

- const string& num: Число (возможно, с минусом).

**Выход:**

- string: Число без знака.

```

1 string BigArithmeticCalc::removeSign(const string& num) const {
2     if (isNegative(num)) {
3         return num.substr(1);
4     }
5     return num;
6 }

```

Listing 10: Реализация removeSign()

#### 2.4.4. Метод addSign(const string& num, bool negative)

**Назначение:** Добавляет знак минус к числу, если указан флаг отрицательности. Нейтральный элемент по сложению всегда возвращается без знака.

**Вход:**

- const string& num: Число без знака.
- bool negative: Флаг отрицательности.

**Выход:**

- string: Число со знаком (если negative = true) или без знака.

```
1 string BigArithmeticCalc::addSign(const string& num, bool
   negative) const {
2     if (num == small.getAdditiveIdentity()) return num;
3     return negative ? "-" + num : num;
4 }
```

Listing 11: Реализация addSign()

#### 2.4.5. Метод isValidNumber(const string& num)

**Назначение:** Проверяет, что все символы числа принадлежат алфавиту системы.

**Вход:**

- const string& num: Число для проверки.

**Выход:**

- bool: true, если все символы валидны, иначе false.

```
1 bool BigArithmeticCalc::isValidNumber(const string& num) const {
2     if (num.empty()) return false;
3
4     string unsign_num = removeSign(num);
5
6     for (char c : unsign_num) {
7         if (!small.isValidElement(string(1, c))) return false;
8     }
9     return true;
10 }
```

Listing 12: Реализация isValidNumber()

### 2.4.6. Метод isOverflow(const string& num)

**Назначение:** Проверяет, не превышает ли количество разрядов числа максимально допустимое значение (MAX\_DIGITS = 8).

**Вход:**

- const string& num: Число для проверки.

**Выход:**

- bool: true, если произошло переполнение, иначе false.

```
1 bool BigArithmeticCalc::isOverflow(const string& num) const {  
2     string unsign_num = removeSign(num);  
3     string withoutZeros = deleteTrashZeros(unsign_num);  
4     return withoutZeros.length() >  
5         static_cast<size_t>(small.getMaxDigits());  
}
```

Listing 13: Реализация isOverflow()

### 2.4.7. Метод deleteTrashZeros(const string& num)

**Назначение:** Удаляет ведущие нули из числа. Если число состоит только из нулей, возвращает нейтральный элемент по сложению.

**Вход:**

- const string& num: Число с возможными ведущими нулями.

**Выход:**

- string: Число без ведущих нулей.

```
1 string BigArithmeticCalc::deleteTrashZeros(const string& num)
2     const {
3         if (num.empty()) return small.getAdditiveIdentity();
4
5         bool negative = isNegative(num);
6         string unsign_num = removeSign(num);
7
8         size_t firstNonZero = 0;
9         while (firstNonZero < unsign_num.length() &&
10              string(1, unsign_num[firstNonZero]) ==
11                  small.getAdditiveIdentity()) {
12             firstNonZero++;
13         }
14
15         if (firstNonZero == unsign_num.length()) {
16             return small.getAdditiveIdentity();
17         }
18
19         string result = unsign_num.substr(firstNonZero);
20         return addSign(result, negative);
21     }
```

Listing 14: Реализация deleteTrashZeros()

Алгоритм:

1. Удаляется знак (если есть);
2. Подсчитывается количество ведущих нулей;
3. Если все символы — нули, возвращается нейтральный элемент;
4. Иначе возвращается подстрока без ведущих нулей со знаком (если необходимо).

## 2.4.8. Метод `compareBigUnsigned(const string& a, const string& b)`

**Назначение:** Сравнивает два беззнаковых многозначных числа.

**Вход:**

- `const string& a`: Первое число.
- `const string& b`: Второе число.

**Выход:**

- `int`: 0 (если  $a = b$ ), 1 (если  $a > b$ ), -1 (если  $a < b$ ).

```
1 int BigArithmeticCalc:: compareBigUnsigned(const string& a, const
  string& b) const {
2     string normA = deleteTrashZeros(a);
3     string normB = deleteTrashZeros(b);
4
5     if (normA.length() > normB.length()) return 1;
6     if (normA.length() < normB.length()) return -1;
7
8     for (size_t i = 0; i < normA.length(); ++i) {
9         int cmp = small.compareElems(string(1, normA[i]),
10                                     string(1, normB[i]));
11         if (cmp != 0) return cmp;
12     }
13     return 0;
14 }
```

Listing 15: Реализация `compareBigUnsigned()`

Алгоритм:

1. Нормализация чисел (удаление ведущих нулей);
2. Сравнение по количеству разрядов;
3. При равной длине — посимвольное сравнение слева направо.

## Операции большой арифметики

### 2.4.9. Метод `addBigUnsigned(const string& a, const string& b)`

**Назначение:** Выполняет сложение двух беззнаковых многозначных чисел столбиком с учётом переноса между разрядами.

**Вход:**

- `const string& a`: Первое слагаемое.
- `const string& b`: Второе слагаемое.

**Выход:**

- `string`: Сумма  $a + b$  без ведущих нулей.

```
1 string BigArithmeticCalc::addBigUnsigned(const string& a, const
  string& b) const {
2     string result = "";
3     string carry = small.getAdditiveIdentity();
4
5     int i = a.length() - 1;
6     int j = b.length() - 1;
7
8     while (i >= 0 || j >= 0 || carry !=
  small.getAdditiveIdentity()) {
9         string digit_a = (i >= 0) ? string(1, a[i]) :
  small.getAdditiveIdentity();
10        string digit_b = (j >= 0) ? string(1, b[j]) :
  small.getAdditiveIdentity();
11
12        auto [sum, new_carry] = small.addWithCarry(digit_a,
  digit_b, carry);
13
14        result = sum + result;
15        carry = new_carry;
16
17        i--;
18        j--;
19    }
20
21    return deleteTrashZeros(result);
22 }
```

Listing 16: Реализация `addBigUnsigned()`

Алгоритм выполняет сложение справа налево (от младших разрядов к старшим), используя таблицу `additionTableWithCarry` для учёта переносов.

### 2.4.10. Метод `subtractBigUnsigned(const string& a, const string& b)`

**Назначение:** Выполняет вычитание двух беззнаковых многозначных чисел столбиком с учётом заимствования из старших разрядов. Требуется, чтобы  $a \geq b$ .

### Вход:

- `const string& a`: Уменьшаемое ( $a \geq b$ ).
- `const string& b`: Вычитаемое.

### Выход:

- `string`: Разность  $a - b$  без ведущих нулей.

```
1 string BigArithmeticCalc::subtractBigUnsigned(const string& a,
2 const string& b) const {
3     if (compareBigUnsigned(a, b) < 0) {
4         throw runtime_error("subtraction: a < b");
5     }
6
7     string result = "";
8     string borrow = small.getAdditiveIdentity();
9
10    int i = a.length() - 1;
11    int j = b.length() - 1;
12
13    while (i >= 0) {
14        string digit_a = string(1, a[i]);
15        string digit_b = (j >= 0) ? string(1, b[j]) :
16            small.getAdditiveIdentity();
17
18        digit_a = small.smallSubtract(digit_a, borrow);
19
20        string diff;
21        if (small.compareElems(digit_a, digit_b) >= 0) {
22            diff = small.smallSubtract(digit_a, digit_b);
23            borrow = small.getAdditiveIdentity();
24        } else {
25            string augmented = small.addByHasse(digit_a,
26                small.getMaxSymbol());
27            diff = small.smallSubtract(augmented, digit_b);
28            borrow = small.nextElem(small.getAdditiveIdentity());
29        }
30
31        result = diff + result;
32        i--;
33        j--;
34    }
35
36    return deleteTrashZeros(result);
37 }
```

Listing 17: Реализация `subtractBigUnsigned()`

Алгоритм выполняет вычитание справа налево с учётом заимствования из старших разрядов.

### 2.4.11. Метод `multiplyBigUnsigned(const string& a, const string& b)`

**Назначение:** Выполняет умножение двух беззнаковых многозначных чисел столбиком. Число  $a$  умножается на каждую цифру числа  $b$ , промежуточные результаты сдвигаются и суммируются.

**Вход:**

- `const string& a`: Множимое.
- `const string& b`: Множитель.

**Выход:**

- `string`: Произведение  $a \times b$  без ведущих нулей.

```
1 string BigArithmeticCalc::multiplyBigUnsigned(const string& a,  
2     const string& b) const {  
3     string result = small.getAdditiveIdentity();  
4  
5     for (int i = b.length() - 1; i >= 0; --i) {  
6         string digit = string(1, b[i]);  
7         string partial = multiplyByDigit(a, digit);  
8  
9         int shift = b.length() - 1 - i;  
10        for (int k = 0; k < shift; ++k) {  
11            partial += small.getAdditiveIdentity();  
12        }  
13        result = addBigUnsigned(result, partial);  
14    }  
15  
16    return deleteTrashZeros(result);  
17 }
```

Listing 18: Реализация `multiplyBigUnsigned()`

Алгоритм:

1. Умножение числа  $a$  на каждую цифру числа  $b$ ;
2. Сдвиг промежуточных результатов влево на соответствующее количество разрядов;
3. Суммирование всех промежуточных результатов.

### 2.4.12. Метод `multiplyByDigit(const string& num, const string& digit)`

**Назначение:** Умножает многозначное число на однозначное число (один элемент алфавита). Используется в методе `multiplyBigUnsigned`.

**Вход:**



- `const string& num`: Многочисленное число.
- `const string& digit`: Однозначное число (элемент алфавита).

### Выход:

- `string`: Произведение  $num \times digit$ .

```

1 string BigArithmeticCalc::multiplyByDigit(const string& num,
2                                           const string& digit)
3                                           const {
4     if (digit == small.getAdditiveIdentity()) {
5         return small.getAdditiveIdentity();
6     }
7
8     string result = "";
9     string carry = small.getAdditiveIdentity();
10
11     for (int i = num.length() - 1; i >= 0; --i) {
12         string current_digit = string(1, num[i]);
13         string product = small.multiplyByHasse(current_digit,
14         digit);
15         product = small.addByHasse(product, carry);
16
17         if (product.length() > 1) {
18             carry = string(1, product[0]);
19             result = string(1, product[1]) + result;
20         } else {
21             result = product + result;
22             carry = small.getAdditiveIdentity();
23         }
24     }
25
26     if (carry != small.getAdditiveIdentity()) {
27         result = carry + result;
28     }
29
30     return result;
31 }

```

Listing 19: Реализация `multiplyByDigit()`

### 2.4.13. Метод `divideBigUnsigned(const string& a, const string& b)`

**Назначение:** Выполняет деление двух беззнаковых многозначных чисел ”уголком”. Возвращает частное и остаток.

**Вход:**

- `const string& a`: Делимое.
- `const string& b`: Делитель ( $b \neq 0$ ).

**Выход:**

- `pair<string, string>`: Пара (частное, остаток).

```
1 pair<string, string> BigArithmeticCalc::divideBigUnsigned(const
  string& a const string& b) const {
2   if (b == small.getAdditiveIdentity()) {
3     throw runtime_error("division by zero");
4   }
5
6   if (compareBigUnsigned(a, b) < 0) {
7     return {small.getAdditiveIdentity(), a};
8   }
9
10  string quotient = "";
11  string remainder = "";
12
13  for (size_t i = 0; i < a.length(); ++i) {
14    remainder += a[i];
15    remainder = deleteTrashZeros(remainder);
16
17    string count = small.getAdditiveIdentity();
18    while (compareBigUnsigned(remainder, b) >= 0) {
19      remainder = subtractBigUnsigned(remainder, b);
20      count = small.nextElem(count);
21    }
22
23    quotient += count;
24  }
25
26  return {deleteTrashZeros(quotient),
27          deleteTrashZeros(remainder)};
28 }
```

Listing 20: Реализация `divideBigUnsigned()`

Алгоритм выполняет деление ”уголком”, последовательно вычитая делитель из текущего остатка.

## Публичные операции с учётом знаков

### 2.4.14. Метод `add(const string& a, const string& b)`

**Назначение:** Выполняет сложение двух чисел с учётом знаков. Обрабатывает случаи сложения чисел с одинаковыми и разными знаками.

**Вход:**

- `const string& a`: Первое слагаемое.
- `const string& b`: Второе слагаемое.

**Выход:**

- `string`: Сумма  $a + b$  со знаком.

```
1 string BigArithmeticCalc::add(const string& a, const string& b)
2     const {
3         if (! isValidNumber(a) || ! isValidNumber(b)) {
4             throw runtime_error("invalid number format");
5         }
6
7         bool neg_a = isNegative(a);
8         bool neg_b = isNegative(b);
9
10        string abs_a = removeSign(a);
11        string abs_b = removeSign(b);
12
13        string result;
14        bool result_negative;
15
16        if (neg_a == neg_b) {
17            result = addBigUnsigned(abs_a, abs_b);
18            result_negative = neg_a;
19        } else {
20            int cmp = compareBigUnsigned(abs_a, abs_b);
21            if (cmp >= 0) {
22                result = subtractBigUnsigned(abs_a, abs_b);
23                result_negative = neg_a;
24            } else {
25                result = subtractBigUnsigned(abs_b, abs_a);
26                result_negative = neg_b;
27            }
28        }
29
30        if (isOverflow(result)) {
31            throw runtime_error("overflow: result exceeds
32                                MAX_DIGITS");
33        }
34
35        return addSign(result, result_negative);
36    }
```

---

### Listing 21: Реализация add()

Алгоритм:

1. Проверка валидности чисел;
2. Извлечение знаков и модулей;
3. Если знаки одинаковые — складываются модули;
4. Если знаки разные — вычитается меньший модуль из большего;
5. Проверка переполнения;
6. Добавление знака к результату.

#### 2.4.15. Метод `subtract(const string& a, const string& b)`

**Назначение:** Выполняет вычитание двух чисел с учётом знаков. Сводится к сложению с противоположным числом:  $a - b = a + (-b)$ .

**Вход:**

- `const string& a`: Уменьшаемое.
- `const string& b`: Вычитаемое.

**Выход:**

- `string`: Разность  $a - b$  со знаком.

```
1 string BigArithmeticCalc::subtract(const string& a, const string&
   b) const {
2     string neg_b = isNegative(b) ? removeSign(b) : "-" + b;
3     return add(a, neg_b);
4 }
```

### Listing 22: Реализация subtract()

#### 2.4.16. Метод `multiply(const string& a, const string& b)`

**Назначение:** Выполняет умножение двух чисел с учётом знаков. Знак результата определяется правилом:  $(-) \times (-) = (+)$ ,  $(+) \times (-) = (-)$ ,  $(+) \times (+) = (+)$ .

**Вход:**

- `const string& a`: Множимое.
- `const string& b`: Множитель.

### Выход:

- string: Произведение  $a \times b$  со знаком.

```
1 string BigArithmeticCalc::multiply(const string& a, const string&
  b) const {
2     if (!isValidNumber(a) || !isValidNumber(b)) {
3         throw runtime_error("invalid number format");
4     }
5
6     bool neg_a = isNegative(a);
7     bool neg_b = isNegative(b);
8
9     string abs_a = removeSign(a);
10    string abs_b = removeSign(b);
11
12    string result = multiplyBigUnsigned(abs_a, abs_b);
13
14    if (isOverflow(result)) {
15        throw runtime_error("overflow: result exceeds MAX_DIGITS");
16    }
17
18    bool result_negative = (neg_a != neg_b);
19
20    return addSign(result, result_negative);
21 }
```

Listing 23: Реализация multiply()

### 2.4.17. Метод divide(const string& a, const string& b)

**Назначение:** Выполняет деление двух чисел с учётом знаков. Возвращает только частное (остаток не выводится). Знак результата определяется правилом умножения.

#### Вход:

- const string& a: Делимое.
- const string& b: Делитель ( $b \neq 0$ ).

#### Выход:

- string: Частное  $a/b$  со знаком.

```
1 string BigArithmeticCalc:: divide(const string& a, const string&
  b) const {
2     if (!isValidNumber(a) || !isValidNumber(b)) {
3         throw runtime_error("invalid number format");
4     }
5
6     if (removeSign(b) == small.getAdditiveIdentity()) {
```

```

7         throw runtime_error("division by zero");
8     }
9
10    bool neg_a = isNegative(a);
11    bool neg_b = isNegative(b);
12
13    string abs_a = removeSign(a);
14    string abs_b = removeSign(b);
15
16    auto [quotient, remainder] = divideBigUnsigned(abs_a, abs_b);
17
18    bool result_negative = (neg_a != neg_b);
19
20    return addSign(quotient, result_negative);
21 }

```

Listing 24: Реализация divide()

## 3. Результаты работы программы

### 3.1. Запуск программы

При запуске программа выводит приветственное сообщение и информацию о командах. Пользователь может вводить команды или арифметические выражения.

```
===== Калькулятор БОЛЬШОЙ конечной арифметики Z8 =====  
  
Введите 'help' для справки по командам  
Введите 'info' для информации о системе  
  
===== ИНФОРМАЦИЯ О КОНЕЧНОЙ АРИФМЕТИКЕ =====  
  
размерность: Z8  
аддитивная единица (0): 'a'  
мультипликативная единица (1): 'b'  
  
максимальное количество разрядов: 8  
диапазон чисел: [-hhhhhhhh; hhhhhhhh]  
  
специальные правила:  
•  $a * a = [-hhhhhhhh; hhhhhhhh]$  (универсум)  
•  $\text{число} / a = \emptyset$  (пустое множество)  
•  $a / a = [-hhhhhhhh; hhhhhhhh]$  (диапазон всех чисел)  
• переполнение ( $> 8$  разрядов) = ERR: overflow  
===== ДИАГРАММА ХАССЕ (правило +1) =====  
  
Отношение порядка элементов Z8:  
  
a => b => c => e => d => g => f => h => a (цикл)  
  
=====
```

Рис. 3: Приветственное сообщение при запуске программы

## 3.2. Сценарий 1: Просмотр справки

### 3.2.1. Команда help

При вводе команды `help` программа выводит список всех доступных команд и их описание.

```
===== ПОМОЩЬ =====

доступные команды:

БОЛЬШАЯ АРИФМЕТИКА (до 8 разрядов, с отрицательными):
<число> + <число>      - сложение больших чисел
<число> - <число>      - вычитание больших чисел
<число> * <число>      - умножение больших чисел
<число> / <число>      - деление больших чисел

примечание: отрицательные числа пишутся с '-' (например: -abc)

ИНФОРМАЦИЯ:
info                - информация об арифметике
hasse               - диаграмма хассе
tables              - все таблицы операций
add_table           - таблица сложения
mul_table           - таблица умножения
sub_table           - таблица вычитания
div_table           - таблица деления
help                - эта справка
exit / quit         - выход

примеры:
abd + ghf           - сложение положительных
-abd + ghf          - сложение отрицательного и положительного
hhhhhhh + b         - переполнение (ERR: overflow)
-c / d              - деление отрицательного на положительное
a / a               - результат: [-hhhhhhh; hhhhhhhh]
g / a               - результат: ∅ (пустое множество)

особенности деления с остатком:
• при делении -a / b: частное увеличивается на 1, остаток = b - r
• остаток может быть многозначным

=====
```

Рис. 4: Результат выполнения команды `help`

### 3.2.2. Команда info

При вводе команды `info` программа выводит информацию о конфигурации системы.

```
===== ИНФОРМАЦИЯ О КОНЕЧНОЙ АРИФМЕТИКЕ =====

размерность: Z8
аддитивная единица (0): 'a'
мультипликативная единица (1): 'b'

максимальное количество разрядов: 8
диапазон чисел: [-hhhhhhh; hhhhhh]

специальные правила:
• a * a = [-hhhhhhh; hhhhhh] (универсум)
• число / a = ∅ (пустое множество)
• a / a = [-hhhhhhh; hhhhhh] (диапазон всех чисел)
• переполнение (> 8 разрядов) = ERR: overflow
```

Рис. 5: Результат выполнения команды `info`



### 3.3. Сценарий 2: Просмотр диаграммы Хассе

#### 3.3.1. Команда `hasse`

При вводе команды `hasse` программа выводит диаграмму Хассе — визуальное представление отношения порядка на алфавите.

```
===== ДИАГРАММА ХАССЕ (правило +1) =====  
  
Отношение порядка элементов Z8:  
  
a => b => c => e => d => g => f => h => a (цикл)  
  
=====
```

Рис. 6: Диаграмма Хассе

Диаграмма показывает циклический порядок элементов алфавита. Каждый элемент связан со следующим по правилу ”+1”.

### 3.4. Сценарий 3: Операции с большими числами

#### 3.4.1. Сложение положительных чисел

Пользователь вводит выражение для сложения двух положительных чисел.

```
calc> abc + def  
  
===== Результат большой арифметики =====  
abc + def = dga  
=====
```

Рис. 7: Сложение положительных чисел

Результат вычисляется столбиком с учётом переносов между разрядами.

#### 3.4.2. Сложение с переносами

Пример, демонстрирующий механизм переноса в старшие разряды.

```
calc> hhhh + b  
  
===== Результат большой арифметики =====  
hhhh + b = baaaa  
=====
```

Рис. 8: Сложение с переносом

При сложении `f + b` происходит переход через границу алфавита, что вызывает перенос в следующий разряд.

### 3.4.3. Сложение с отрицательными числами

Пример сложения чисел с разными знаками.

```
calc> abc + -abc

===== Результат большой арифметики =====
  abc + -abc = a
=====

calc> -def + bcd

===== Результат большой арифметики =====
 -def + bcd = -ebc
=====

calc> -bcd + def

===== Результат большой арифметики =====
 -bcd + def = ebc
=====
```

Рис. 9: Сложение с отрицательными числами

Программа правильно обрабатывает случаи:

- Сложение противоположных чисел (результат — нейтральный элемент)
- Сложение чисел с разными знаками (вычитание модулей)

### 3.4.4. Вычитание положительных чисел

```
calc> def - abc

===== Результат большой арифметики =====
  def - abc = dcd
=====

calc> abc - def

===== Результат большой арифметики =====
  abc - def = -dcd
=====
```

Рис. 10: Вычитание положительных чисел

При вычитании меньшего из большего результат положительный, при вычитании большего из меньшего — отрицательный.

### 3.4.5. Вычитание с заимствованием

Пример, демонстрирующий механизм заимствования из старших разрядов.

```

calc> baaa - hhh

===== Результат большой арифметики =====
      baaa - hhh = b
=====

```

Рис. 11: Вычитание с заимствованием

При вычитании  $a - b$  происходит заимствование из старшего разряда.

### 3.4.6. Умножение положительных чисел

```

calc> abc * h

===== Результат большой арифметики =====
      abc * h = baf
=====

calc> bc * de

===== Результат большой арифметики =====
      bc * de = gef
=====

```

Рис. 12: Умножение положительных чисел

Умножение выполняется столбиком с промежуточным суммированием.

### 3.4.7. Умножение с отрицательными числами

```

calc> -bc * de

===== Результат большой арифметики =====
      -bc * de = -gef
=====

calc> -abc * -h

===== Результат большой арифметики =====
      -abc * -h = baf
=====

calc> abc * -h

===== Результат большой арифметики =====
      abc * -h = -baf
=====

```

Рис. 13: Умножение с отрицательными числами

Знак результата определяется правилом:

- $+$   $\times$   $+$   $=$   $+$
- $-$   $\times$   $-$   $=$   $+$
- $+$   $\times$   $-$   $=$   $-$
- $-$   $\times$   $+$   $=$   $-$

### 3.4.8. Деление положительных чисел

```
calc> gef / bc

===== Результат большой арифметики =====
gef / bc = Q: de | R: a
=====

calc> abcd / ef

===== Результат большой арифметики =====
abcd / ef = Q: c | R: ea
=====
```

Рис. 14: Деление положительных чисел

Деление выполняется ”уголком” с получением целой части.

### 3.4.9. Деление с отрицательными числами

```
calc> -gef / bc

===== Результат большой арифметики =====
-gef / bc = Q: -de | R: a
=====

calc> gef / -bc

===== Результат большой арифметики =====
gef / -bc = Q: -de | R: a
=====

calc> -gef / -bc

===== Результат большой арифметики =====
-gef / -bc = Q: de | R: a
=====
```

Рис. 15: Деление с отрицательными числами

## 3.5. Сценарий 4: Обработка ошибок

### 3.5.1. Некорректный формат числа

При вводе числа, содержащего символы вне алфавита, программа выводит сообщение об ошибке.

```
calc> abx / f

===== Результат большой арифметики =====
abx / f = ERR
=====
```

Рис. 16: Ошибка: некорректный формат числа

Символ x не принадлежит алфавиту системы.

### 3.5.2. Деление на ноль

При попытке деления на нейтральный элемент по сложению (аналог нуля) программа выводит ошибку.

```
calc> abc / a

===== Результат большой арифметики =====
abc / a = Q: ∅ | R: ∅
=====
```

Рис. 17: Ошибка: деление на ноль

### 3.5.3. Переполнение разрядов

При превышении максимального количества разрядов ( $\text{MAX\_DIGITS} = 8$ ) программа выводит ошибку переполнения.

```
calc> hhhhhhhhhhhh * hh

===== Результат большой арифметики =====
hhhhhhhhhh * hh = ERR: overflow
=====
```

Рис. 18: Ошибка: переполнение

Результат операции превышает 8 разрядов, что не допускается системой.

### 3.5.4. Незвестная команда

При вводе неизвестной команды программа выводит сообщение об ошибке.

```
calc> home  
Ошибка: неверный формат команды. Введите 'help' для справки
```

Рис. 19: Ошибка: неизвестная команда

### 3.6. Выход из программы

При вводе команды `exit` или `quit` программа завершает работу с прощальным сообщением.

```
calc> home  
Ошибка: неверный формат команды. Введите 'help' для справки
```

Рис. 20: Завершение работы программы

## Заключение

В ходе курсовой работы был разработан калькулятор большой конечной арифметики  $\langle Z_8^8; +, * \rangle$  с поддержкой восьми разрядов. Калькулятор выполняет четыре арифметические операции: сложение, вычитание, умножение и деление над многозначными числами в заданной системе счисления.

## Основные результаты

В программной части была реализована малая конечная арифметика с операциями над однозначными элементами через диаграмму Хассе. Построены таблицы операций малой арифметики для сложения, умножения, вычитания и деления. Для выполнения операций столбиком создана таблица сложения с переносом, а для операции деления построена таблица обратных элементов.

На основе малой арифметики реализована большая арифметика с операциями над многозначными числами до восьми разрядов. Добавлена поддержка отрицательных чисел с корректной обработкой знаков при выполнении всех операций.

Разработана система команд для просмотра информации о системе, диаграммы Хассе и таблиц операций. Реализована обработка ошибок, включая деление на ноль, переполнение разрядов и некорректный формат чисел.

Реализованные алгоритмы показали корректность работы для различных входных данных.

## Достоинства реализации

Достоинством программы является модульная архитектура: малая арифметика работает с однозначными элементами, а большая арифметика использует её для операций столбиком. Код разделён по файлам, каждый из которых отвечает за свою часть функционала.

Использование предвычисленных таблиц операций делает вычисления эффективными: все операции малой арифметики вычисляются один раз при инициализации, а затем используются для быстрого доступа к значениям. Это позволяет избежать повторных вычислений при работе с многозначными числами.

## Недостатки реализации

Конфигурация системы (алфавит, правило "+1", нейтральные элементы) задаётся в файле `config.hpp` и требует перекомпиляции при изменении. Пользователь не может менять параметры во время работы программы.

Также программа ограничена восьмью разрядами, что не позволяет работать с очень большими числами. При превышении лимита возникает ошибка переполнения.

## **Возможности масштабирования и улучшения**

Все операции выполняются через консольный интерфейс. Визуализация таблиц и диаграммы Хассе была бы удобнее через GUI при помощи фреймворка Qt.

Можно добавить возможность загружать конфигурацию из файла, чтобы пользователь мог менять алфавит и правило "+1" без повторной компиляции программы.



## Список литературы

- [1] Сайт кафедры с учебными материалами по курсу «Дискретная математика». Ссылка: <https://tema.spbstu.ru/dismath/> (Дата обращения: 21.10.2025).
- [2] Новиков Ф.А. *Дискретная математика*. Учебник. Ссылка на PDF: <https://stugum.wordpress.com/wp-content/uploads/2014/03/novikov.pdf> (Дата обращения: 21.10.2025).