

Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта
02.03.01 Математика и компьютерные науки

Отчёт по курсовой работе
по дисциплине
Дискретная математика
«Калькулятор большой конечной арифметики»

Выполнил:

студент группы 5130201/40003

Джабраилов А.В.

Принял:

Востров А.В.

«_____» _____ 2025 г.

Санкт-Петербург, 2025

Содержание

Введение	3
1 Постановка задачи	4
2 Математическое описание	5
2.1 Малая конечная арифметика	5
2.2 Большая конечная арифметика	8
2.3 Арифметические действия	10
2.4 Частные случаи	14
3 Особенности реализации	15
3.1 Реализация логики арифметики	15
3.2 Реализация консольного меню	27
4 Результаты работы программы	28
Заключение	32
Список литературы	35
Приложение 1. Файл Z8.h	36
Приложение 2. Файл Z8.cpp	38
Приложение 3. Файл main.cpp	50

Введение

Актуальность и цель работы

Конечные алгебраические структуры, такие как кольца, играют фундаментальную роль в дискретной математике. Понимание принципов построения таких систем и выполнения действий в них без привлечения иных арифметик позволяет глубже понять природу вычислений и пользоваться изолированными от других систем арифметиками.

В рамках данной работы рассматривается конечная арифметика Z_8 с заданными алгебраическими свойствами, такими как коммутативность, дистрибутивность и др. Цель работы — составить символьный калькулятор данной арифметики, основываясь только на "малой" конечной арифметике, заданной таблицей $+1$. Это моделирует ситуацию, в которой вычисления должны происходить внутри изолированной системы без возможности перевода в иные, более привычные для использования.

1 Постановка задачи

В рамках лабораторной работы необходимо реализовать калькулятор "большой" конечной арифметики $\langle Z_i; +, * \rangle$ (8 разрядов) для четырех действий $(+, -, *, /)$ на основе "малой" конечной арифметики, где задано правило «+1» и выполняются свойства коммутативности $(+, *)$, ассоциативности $(+, *)$, дистрибутивности $*$ относительно $+$, заданы аддитивная единица «а» и мультипликативная единица «б», а также выполняется свойство: $(\forall x) * a = a$.

Необходимо реализовать программу, которая:

- поддерживает все четыре действия, а именно:
 - сложение,
 - вычитание,
 - умножение,
 - деление, в т.ч. с остатком,
- поддерживает числа до 8 разрядов включительно (например, возможно выражение $bbb + ccc$),
- поддерживает отрицательные числа,
- выполняет вычисления только в рамках данной арифметики без перевода в иные системы счисления.

2 Математическое описание

2.1 Малая конечная арифметика

Рассмотрим малую конечную арифметику

$$Z_8 = \{a, b, c, d, e, f, g, h\},$$

в которой задана таблица «+1», определяющая правило перехода от элемента к следующему:

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

Её алгебраическая структура —

$$\langle Z_8; +, -, \cdot, / \rangle.$$

Отношение порядка (диаграмма Хассе) имеет следующий вид:

$$a \Rightarrow b \Rightarrow d \Rightarrow h \Rightarrow c \Rightarrow e \Rightarrow g \Rightarrow f.$$

У данной арифметики есть следующие свойства:

- элемент a является **аддитивной единицей**:

$$\forall x \in Z_8 \quad x + a = x;$$

- свойство **поглощения**:

$$\forall x \in Z_8 \quad x \cdot a = a;$$

- элемент b является **мультипликативной единицей**:

$$\forall x \in Z_8 \quad x \cdot b = x;$$

- **коммутативность:**

$$\forall a, b \in Z_8 \quad a \cdot b = b \cdot a,$$

$$\forall a, b \in Z_8 \quad a + b = b + a;$$

- **ассоциативность:**

$$\forall a, b, c \in Z_8 \quad (a \cdot b) \cdot c = a \cdot (b \cdot c);$$

- **дистрибутивность:**

$$(a + b) \cdot c = a \cdot c + b \cdot c.$$

Коммутативность и наличие мультипликативной единицы означают, что данная арифметика — это коммутативное кольцо с единицей.

Для данной арифметики были составлены таблицы сложения и умножения.

Сложение

Таблица сложения строилась на основе диаграммы Хассе и свойств данной арифметики. Примеры вычислений:

Пример 1: $a + d$. Так как a по условию аддитивная единица, $a + d = d$.

Пример 2: $g + h$. Расстояние между h и a (аддитивной единицей) на диаграмме Хассе равно трём, следует, что для получения результата арифметического действия нужно по диаграмме Хассе сделать три шага от g вправо. Получаем следующую последовательность: $g \Rightarrow f \Rightarrow a \Rightarrow b$. Итого: $g + h = b$.

Полученная таблица изображена ниже [1]:

Таблица 1: Таблица сложения для арифметики Z_8 .

.	a	b	d	h	c	e	g	f
a	a	b	d	h	c	e	g	f
b	b	d	h	c	e	g	f	a
d	d	h	c	e	g	f	a	b
h	h	c	e	g	f	a	b	d
c	c	e	g	f	a	b	d	h
e	e	g	f	a	b	d	h	c
g	g	f	a	b	d	h	c	e
f	f	a	b	d	h	c	e	g

Умножение

Таблица умножения строилась на основе диаграммы Хассе и свойств данной арифметики, а также путём преобразования множителей до суммы чисел b (мультипликативная единица), так как по ней вычислять произведения проще всего. Примеры вычислений:

Пример 1: $a \cdot d$. Так как для a по условию выполняется свойство поглощения, $a \cdot d = a$.

Пример 2: $g \cdot h$. Расстояние между h и b (мультипликативной единицей) на диаграмме Хассе равно двум, следует, что h больше b в три раза, $\rightsquigarrow h = b + b + b$. Таким образом, можно преобразовать исходное выражение следующим образом:

$$g \cdot h = g \cdot (b + b + b) = g \cdot b + g \cdot b + g \cdot b = g + g + g = c + g = d.$$

Итоговое значение — d .

Полученная таблица изображена ниже [2]:

Таблица 2: Таблица умножения для арифметики Z_8 .

.	a	b	d	h	c	e	g	f
a	a	a	a	a	a	a	a	a
b	a	b	b	d	h	c	e	f
d	a	d	c	g	a	d	c	g
h	a	h	g	b	c	f	d	e
c	a	c	a	c	a	c	a	c
e	a	e	d	f	c	b	g	h
g	a	g	c	d	a	g	c	d
f	a	f	g	e	c	h	d	b

2.2 Большая конечная арифметика

Так как необходимо реализовать калькулятор большой конечной арифметики на базе малой с поддержкой чисел до 8 разрядов включительно, алгебраическая структура данной арифметики следующая:

$$\langle Z_8^8; +, -, *, / \rangle .$$

Были построены таблицы переносов по сложению и переносов по умножению для этой арифметики.

Переносы по сложению

Значение переноса по сложению вычислялось по количеству переходов $f \Rightarrow a$ по диаграмме Хассе, поскольку это замыкает кольцо и означает переход от наибольшего элемента к наименьшему.

Пример 1: $b + d$. При решении данного примера не происходит перехода по диаграмме Хассе от f к a , так как переходы выглядят следующим образом:

$$b \Rightarrow d \Rightarrow h.$$

Так как нет переходов $f \Rightarrow a$, перенос по данному сложению равен a .

Пример 2: $f + h$. Переходы по диаграмме Хассе для данного примера выглядят следующим образом:

$$f \Rightarrow a \Rightarrow b \Rightarrow d.$$

Так как есть один переход $f \Rightarrow a$, перенос по данному сложению равен b .

Полученная таблица переносов по сложению изображена ниже [3]:

Таблица 3: Таблица переносов по сложению для арифметики Z_8 .

.	a	b	d	h	c	e	g	f
a	a	a	a	a	a	a	a	a
b	a	a	a	a	a	a	a	b
d	a	a	a	a	a	a	b	b
h	a	a	a	a	a	b	b	b
c	a	a	a	a	b	b	b	b
e	a	a	a	b	b	b	b	b
g	a	a	a	b	b	b	b	b
f	a	b	b	b	b	b	b	b

Переносы по умножению

Значение переноса по умножению так же вычислялось по количеству переходов $f \Rightarrow a$ по диаграмме Хассе.

Пример 1: $b \cdot d$. При решении данного примера не происходит перехода по диаграмме Хассе от f к a , так как решение выглядит следующим образом:

$$b \cdot d = d b — \text{мультипликативная единица.}$$

Так как нет переходов $f \Rightarrow a$, перенос по данному сложению равен a .

Пример 2: $c \cdot c$. Решение данного примера выглядит следующим образом:

$$c \cdot c = c \cdot (b + b + b + b) = c \cdot b + c \cdot b + c \cdot b + c \cdot b = c + c + c + c,$$

Рассмотрим половину данного выражения, а именно сумму только двух c .

$$c + c = c + b + b + b + b.$$

Переходы по диаграмме Хассе:

$$c \Rightarrow e \Rightarrow g \Rightarrow f \Rightarrow a.$$

$$c + c = a — \text{один переход,}$$

$$\rightsquigarrow c + c + c + c = a + a [\text{два перехода}] = a [\text{дополнительных переходов нет}].$$

Так как есть два перехода $f \Rightarrow a$, перенос по данному умножению равен d .

Полученная таблица переносов по умножению изображена ниже [4]:

2.3 Арифметические действия

Для выполнения действий $(+, -, *, /)$ была составлена следующая логика.

Таблица 4: Таблица переносов по умножению для арифметики Z_8 .

.	a	b	d	h	c	e	g	f
a	a	a	a	a	a	a	a	a
b	a	a	a	a	a	a	a	a
d	a	a	a	a	b	b	b	b
h	a	a	a	b	b	b	d	d
c	a	a	b	b	d	d	h	h
e	a	a	b	b	d	h	h	c
g	a	a	b	d	h	h	c	e
f	a	a	b	d	h	c	e	g

Сложение

Пусть надо сложить числа A и B .

1. Выбирается наибольшее из этих двух чисел. Пусть наибольшее в данном случае A .
2. К A прибавляется элемент b (мультипликативная единица) B раз. Прибавление единицы, в свою очередь, производится по следующей логике.
 - 2.1 Если разряд, к которому прибавлялось b , не был равен f до прибавления, измененным остается только текущий разряд.
 - 2.2 В противном случае, b прибавляется к следующему разряду (перенос), и так до тех пор, пока не надо будет совершать перенос.

Выбор наибольшего из двух чисел в шаге 1 обоснован оптимизацией вычислений, т.к. если мы складываем, например, b и $fffffffg$, намного быстрее

прибавить один раз b к $fffffg$, чем наоборот. Такой подход корректен, т.к. данная арифметика коммутативна.

Умножение

Пусть надо умножить число A на B .

1. Выбирается наибольшее из этих двух чисел (коммутативность). Пусть наибольшее в данном случае A .
2. К A прибавляется A $B-1$ раз, что реализовано через правило «+1».

Выбор наибольшего из двух чисел в шаге 1 так же обоснован оптимизацией вычислений.

Вычитание

В добавок к таблице правила «+1» была создана аналогичная таблица правила «-1».

Логика вычитания реализована так же, как и сложения, за исключением того, что мы "занимаем" разряд в случае, если вычитаемый разряд на момент вычитания равен a , что вызовет переход к f , далее действие производится для старшего разряда, и так до тех пор, пока занимать разряд не придется.

Деление

Деление реализовано так же, как умножение, за исключением того, что вместо сложения мы используем вычитание из числа A числа B до тех пор, пока промежуточный результат вычитания не будет меньше A ; ответ равен текущему результату вычитания и остатку (последнему промежуточному результату).

Отрицательные числа и действия с ними

Операции с отрицательными числами основываются на действиях с положительными, всё заключается в изменении самого действия (например, сложение положительного и отрицательного чисел можно рассматривать как разность), а эти изменения опираются на свойства коммутативных арифметик (например, $-a + b = b + -a = b - a$).

Операции с отрицательными числами проводятся по следующей логике:

Если отрицательно одно число: Пусть модуль отрицательного числа

A , а положительное число B .

1. При сложении вычитаем из модуля большего модуль меньшего; если $A > B$, приписываем минус в начале ответа, в противном случае нет.
2. При вычитании складываем модули чисел.
3. При умножении совершаём умножение и приписываем минус.
4. При делении:
 - 4.1 если остатка при делении нет, приписываем минус;
 - 4.2 если остаток при делении есть, увеличиваем модуль частного на 1 и вычитаем из делителя текущий остаток.

Увеличения модуля в шаге 4.1 не грозит выходом за границы допустимых значений, т.к. в результате такого деления результат не может превышать делимое.

Если оба числа отрицательны: Пусть модули двух отрицательных чисел — A и B соответственно порядку в выражении.

1. При сложении складываем модули и приписываем минус.

2. При вычитании берем модуль большего, вычитаем из него модуль меньшего; если $A > B$, приписываем минус, в противном случае нет.
3. При умножении совершают умножение модулей.
4. При делении совершается деление модулей.

2.4 Частные случаи

В данной алгебре присутствуют следующие частные случаи:

- $\forall x | x \neq a : x/a = \emptyset;$
- $a/a = [-ffffffffff; fffffffffff];$
- если какой-то из членов выражения или результат действия превышает восемь разрядов, это является выходом за допустимые границы.

3 Особенности реализации

3.1 Реализация логики арифметики

Математическая логика программы реализована в классе `Z8Number`. Он содержит в себе поля для хранения данных о числе и функции для проведения действий с ними.

Класс имеет два поля:

- `std::string digits` — модуль числа,
- `bool isNegative` — знак (`true`, если число отрицательно; иначе `false`).

Методы класса перечислены ниже.

- Метод `getNeg`

Вход: `bool isNegative` - отрицательно ли данное число.

Выход: `bool isNegative` - отрицательно ли данное число.

Назначение: возвращает булевое значение, означающее отрицательно ли данное число.

Код:

```
1 bool Z8Number::getNeg() const {  
2     return isNegative;  
3 }
```

- Метод `inc`

Вход: `char c` - текущее число.

Выход: `char c` - следующее число.

Назначение: возвращает число, получающееся в результате действия «+1» над текущим.

Код:

```

1 char Z8Number::inc(char c) {
2     switch (c) {
3         case 'a': return 'b';
4         case 'b': return 'd';
5         case 'c': return 'e';
6         case 'd': return 'h';
7         case 'e': return 'g';
8         case 'f': return 'a';
9         case 'g': return 'f';
10        case 'h': return 'c';
11        default: throw std::invalid_argument("Invalid digit");
12    }
13}

```

- **Метод dec**

Вход: char c - текущее число.

Выход: char c - предыдущее число.

Назначение: возвращает число, получающееся в результате действия «-1» над текущим.

Код:

```

1 char Z8Number::dec(char c) {
2     switch (c) {
3         case 'a': return 'f';
4         case 'b': return 'a';
5         case 'c': return 'h';
6         case 'd': return 'b';
7         case 'e': return 'c';
8         case 'f': return 'g';
9         case 'g': return 'e';
10        case 'h': return 'd';

```

```
11     default: throw std::invalid_argument("Invalid digit");
12 }
13 }
```

- Метод `normalize`

Вход: `std::string s` - введенное число.

Выход: `std::string r` - модуль данного числа без незначащих '0'.

Назначение: возвращает модуль данного числа без незначащих '0'.

Код:

```
1 std::string Z8Number::normalize(const std::string& s) {
2     size_t i = s[0] == '-' ? 1 : 0;
3     while (i < s.length() - 1 && s[i] == '0')
4         ++i;
5     return s.substr(i);
6 }
```

- Метод `isEqual`

Вход: `std::string a, std::string b` - числа для сравнения.

Выход: `bool n` - результат сравнения чисел.

Назначение: сравнивает два числа и возвращает `true`, если их модули равны, иначе `false`.

Код:

```
1 bool Z8Number::isEqual(const std::string& a, const std::string& b) {
2     return normalize(a) == normalize(b);
3 }
```

- Метод `validate`

Вход: `std::string s` - число на проверку корректности.

Выход: `bool n` - корректно ли данное число.

Назначение: проверяет, корректно ли данное число (не превышает ли лимиты и не содержит ли лишних символов).

Код:

```
1 void Z8Number::validate(const std::string& s) {  
2     if (s.empty()) throw std::invalid_argument("Empty number");  
3     if (s.size() > 8 && s[0] != '-' || s.size() > 9) throw std::invalid_argument("More  
4         than 8 digits");  
5     for (char c : s) {  
6         if (c != 'a' && c != 'b' && c != 'c' && c != 'd' &&  
7             c != 'e' && c != 'f' && c != 'g' && c != 'h' && c != '-')  
8             throw std::invalid_argument("Invalid character");  
9     }  
}
```

- **Метод** operator==

Вход: Z8Number current, Z8Number other - числа на сравнение.

Выход: bool n - результат сравнения двух чисел.

Назначение: сравнивает два числа по модулю.

Код:

```
1 bool Z8Number::operator==(const Z8Number& other) const {  
2     return isEqual(digits, other.digits);  
3 }
```

- **Метод** incNumber

Вход: std::string num - число на увеличение.

Выход: std::string new - число после увеличения.

Назначение: увеличивает число на единицу, учитывая переносы.

Код:

```
1 std::string Z8Number::incNumber(const std::string& num) {
```

```

2 std::string res = num;
3 int i = static_cast<int>(res.size()) - 1;
4 while (i >= 0) {
5     char old = res[i];
6     res[i] = inc(old);
7     if (old != 'f') { // no carry
8         break;
9     }
10    if (i == 0) { // carrying until no need to
11        res = 'b' + res;
12        if (res.size() > 8)
13            throw std::overflow_error("Number exceeds 8 digits");
14        break;
15    }
16    --i;
17 }
18 if (normalize(res).size() > 8)
19 throw std::overflow_error("Number exceeds 8 digits");
20 return normalize(res);
21 }
```

- Метод decNumber

Вход: std::string num - число на уменьшение.

Выход: std::string new - число после уменьшения.

Назначение: уменьшает число на единицу, учитывая переносы.

Код:

```

1 std::string Z8Number::decNumber(const std::string& num) {
2     if (isEqual(num, "a"))
3         throw std::domain_error("Cannot decrement zero");
4     std::string res = num;
5     int i = static_cast<int>(res.size()) - 1;
```

```

6   while (i >= 0) {
7       char old = res[i];
8       res[i] = dec(old);
9       if (old != 'a') { // a -> f, so we need to take one from higher
10          break;
11      }
12      --i;
13  }
14  return normalize(res);
15 }
```

- Метод greaterOrEqual

Вход: std::string num1, std::string num2 - числа на сравнение.

Выход: bool n - результат сравнения чисел.

Назначение: сравнивает два числа: больше или равно ли первое, чем второе.

Код:

```

1  bool Z8Number::greaterOrEqual(const std::string& x, const std::string& y) {
2      if (isEqual(x, y)) return true;
3      if (isEqual(y, "a")) return true;
4      if (isEqual(x, "a")) return false;
5      try {
6          subNumbers(x, y);
7          return true;
8      }
9      catch (...) {
10         return false;
11     }
12 }
```

- Метод greater

Вход: std::string num1, std::string num2 - числа на сравнение.

Выход: bool n - результат сравнения чисел.

Назначение: сравнивает два числа: больше ли первое, чем второе.

Код:

```
1 bool greater(const Z8Number& x, const Z8Number& y) {  
2     if (x == y) return false;  
3     if (y == Z8Number("a")) return true;  
4     if (x == Z8Number("a")) return false;  
5     try {  
6         x - y;  
7         return true;  
8     }  
9     catch (...) {  
10        return false;  
11    }  
12 }
```

- **Метод** addNumbers

Вход: std::string num1, std::string num2 - числа для суммирования.

Выход: std::string result - результат суммирования.

Назначение: выполняет сложение двух чисел.

Код:

```
1 std::string Z8Number::addNumbers(const std::string& x, const std::string& y) {  
2     if (isEqual(y, "a")) return x;  
3     std::string res = x;  
4     std::string counter = "a";  
5     while (!isEqual(counter, y)) {  
6         res = incNumber(res);  
7         counter = incNumber(counter);  
8         if (normalize(res).size() > 8)
```

```
9     throw std::overflow_error("Number exceeds 8 digits");
10 }
11 return res;
12 }
```

- Метод subNumbers

Вход: std::string num1, std::string num2 - числа для выполнения вычисления.

Выход: std::string result - разность.

Назначение: выполняет вычитание одного числа из другого, возвращает их разность.

Код:

```
1 std::string Z8Number::subNumbers(const std::string& x, const std::string& y) {
2     if (isEqual(y, "a")) return x;
3     if (isEqual(x, y)) return "a";
4     std::string res = x;
5     std::string counter = "a";
6     while (!isEqual(counter, y)) {
7         res = decNumber(res);
8         counter = incNumber(counter);
9     }
10    return res;
11 }
```

- Метод mulNumbers

Вход: std::string num1, std::string num2 - числа для умножения.

Выход: std::string result - произведение двух чисел.

Назначение: выполняет умножение двух чисел.

Код:

```

1 std::string Z8Number::mulNumbers(const std::string& x, const std::string& y) {
2     if (isEqual(x, "a") || isEqual(y, "a")) return "a";
3     std::string res = x;
4     std::string counter = "b";
5     if (isEqual(y, "ba")) res = res + "a";
6     else if (isEqual(y, "baa")) res = res + "aa";
7     else if (isEqual(y, "baaa")) res = res + "aaa";
8     else if (isEqual(y, "baaaa")) res = res + "aaaa";
9     else if (isEqual(y, "baaaaa")) res = res + "aaaaa";
10    else if (isEqual(y, "baaaaaa")) res = res + "aaaaaa";
11    else if (isEqual(y, "baaaaaaa")) res = res + "aaaaaaa";
12    else {
13        while (!isEqual(counter, y)) {
14            res = addNumbers(res, x);
15            counter = incNumber(counter);
16        }
17    }
18    if (normalize(res).size() > 8)
19        throw std::overflow_error("Number exceeds 8 digits");
20    return res;
21 }
```

- Метод `divNumbers`, `divide`

Вход: `std::string num1, std::string num2` - делимое и делитель.

Выход: `std::string result` - частное.

Назначение: выполняет деление одного числа на другое.

Код:

```

1 std::string Z8Number::divNumbers(const std::string& x, const std::string& y, bool
2 firstIsNeg) {
3     if (isEqual(y, "a")) {
4         if (isEqual(x, "a"))
```

```

4         return "[ffffffffff; ffffffff]";
5     else
6         return "empty set";
7     }
8     if (isEqual(x, "a")) {
9         return "a";
10    }
11
12    std::string quotient = "a";
13    std::string remainder = x;
14
15    while (greaterOrEqual(remainder, y)) {
16        remainder = subNumbers(remainder, y);
17        quotient = incNumber(quotient);
18    }
19
20    if (firstIsNeg && !isEqual(remainder, "a")) {
21        quotient = incNumber(quotient);
22        remainder = subNumbers(y, remainder);
23    }
24
25    if (isEqual(remainder, "a")) {
26        return quotient;
27    }
28    else {
29        return quotient + "(" + remainder + ")";
30    }
31 }
```

- Метод operator +, -, *

Вход: Z8Number num1, Z8Number num2 - числа для проведения арифметического действия.

Выход: Z8Number result - результат выполнения действия.

Назначение: выполнение действия над двумя числами (аналоги вызовов функций).

Код:

```
1 Z8Number Z8Number::operator+(const Z8Number& o) const {  
2     return Z8Number(addNumbers(digits, o.digits));  
3 }  
4 Z8Number Z8Number::operator-(const Z8Number& o) const {  
5     return Z8Number(subNumbers(digits, o.digits));  
6 }  
7 Z8Number Z8Number::operator*(const Z8Number& o) const {  
8     return Z8Number(mulNumbers(digits, o.digits));  
9 }
```

- **Метод operator =**

Вход: Z8Number other - число, к которому надо приравнять текущее.

Выход: Z8Number result - обновленное текущее число.

Назначение: присваивает значение другого числа текущему.

Код:

```
1 Z8Number& Z8Number::operator=(const Z8Number& other) {  
2     digits = other.digits;  
3     isNegative = other.isNegative;  
4     return *this;  
5 }
```

- **Метод toString**

Вход: Z8Number num - текущее число.

Выход: std::string strNum - текущее число в строковом формате.

Назначение: переводит число в строку.

Код:

```
1 std::string Z8Number::toString() const {  
2     return digits;  
3 }
```

- **Метод** calculate

Вход: Z8Number a, Z8Number b, std::string op - числа и знак операции.

Выход: Z8Number result - результат выполнения действия над числами.

Назначение: выполнение действия над двумя числами с учетом знаков и разрядов.

Код:

```
1 void calculate(const Z8Number& a, const Z8Number& b, std::string op) {  
2     std::string result;  
3     if (!(a.isNegative) && !(b.isNegative)) {  
4         try {  
5             if (op == "+") {  
6                 result = (a + b).toString();  
7             }  
8             else if (op == "-") {  
9                 try {  
10                     result = (a - b).toString();  
11                 }  
12                 catch (...) {  
13                     result = "-" + (b - a).toString();  
14                 }  
15             }  
16         }  
17     }  
18 }
```

3.2 Реализация консольного меню

Консольное меню реализовано в файле `main.py`.

Функция `main`

Вход: `int argc, char* argv` – число аргументов командной строки и массив символьных указателей, каждый элемент которого указывает на аргумент командной строки.

Выход: `int n` – значение, указывающее, успешно ли завершилась программа.

Назначение: Точка входа в программу. При запуске программа выводит приветственное сообщение, после чего до каждого ввода пользователя отображает допустимые границы чисел и формат ввода.

Код тела `main`:

```
1 std::cout << "Hello!" << std::endl;
2 std::string s1, s2, op;
3 while (true) {
4     std::cout << "Enter the problem (format: <num> <operation> <num>; nums from -
5         ffffffff to ffffffff):\n>> ";
6     std::cin >> s1;
7     std::cin >> op;
8     std::cin >> s2;
9
10    try {
11        Z8Number a(s1), b(s2);
12        calculate(a, b, op);
13    }
14    catch (...) {
15        std::cerr << "Error: incorrect input" << std::endl;
16    }
```

4 Результаты работы программы

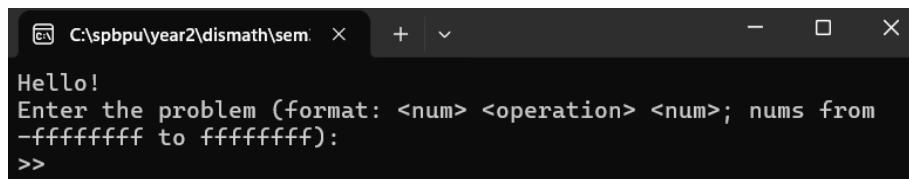
Разработанная программа имеет консольный интерфейс и предоставляет пользователю меню, позволяющее выполнять все действия.

Меню программы

После запуска программы пользователю предлагается ввести пример, который надо решить, в формате <число> <действие> <число>. В случае некорректного ввода (неверный формат, превышение лимитов, некорректные значения чисел) или частных случаев (деление на ноль, превышение лимитов) программа сообщит об этом пользователю.

Пример работы

При запуске программы высвечивается приветственное окно и предложение ввести пример. (рис. 1).



```
C:\spbpu\year2\dismath\sem: x + - ×
Hello!
Enter the problem (format: <num> <operation> <num>; num from
-ffffffff to ffffffff):
>>
```

Рис. 1: Приветственное сообщение при запуске программы.

Когда пользователь введет пример и нажмет Enter, программа выведет результат вычислений.

Сложение приведено на рисунке 2.

```
Enter the problem  
ffffffff):  
>> f + g  
be
```

Рис. 2: Выполнение сложения.

Вычитание приведено на рисунке 3.

```
>> ba - b  
f  
Enter the problem
```

Рис. 3: Выполнение вычитания.

Умножение приведено на рисунке 4.

```
Enter the problem  
ffffffff):  
>> ga * bd  
fca
```

Рис. 4: Выполнение умножения.

Деление приведено на рисунке 5.

```
Enter the problem  
ffffffffff):  
>> -f / h  
-h(d)
```

Рис. 5: Выполнение деления.

Если пользователь сделает некорректный ввод, программа об этом сообщит и предложит ввести пример снова (рис. 6, 8).

```
>> takogo - net  
Error: incorrect input  
Enter the problem (format:
```

Рис. 6: Обработка некорректного пользовательского ввода.

Частные случаи обрабатываются программой корректно (рис. 7, 8, 9).

```
Enter the problem (format: <num>  
>> fffffffff + b  
Error: Number exceeds 8 digits
```

Рис. 7: Обработка попытки выхода за допустимые границы.

```
>> b / a  
empty set
```

Рис. 8: Обработка деления ненулевого числа на `a`.

```
>> a / a  
[-ffffffffff; fffffffffff]  
Enter the problem (fo
```

Рис. 9: Обработка деления `a` на `a`.

Результаты подтвердили, что программа реализована корректно.

Заключение

В ходе лабораторной работы был разработан символьный калькулятор большой конечной арифметики на основе мало конечной арифметики, позволяющий:

- работать с арифметикой Z_8 с элементами (в порядке таблицы «+1») a, b, d, h, c, e, g, f:
 - сложение,
 - разность,
 - умножение,
 - деление,
- работать с числами до 8 разрядов включительно,
- работать с отрицательными числами,
- обрабатывать некорректный ввод пользователя и ограничения в результатах арифметических действий,
- выполнять вычисления только в рамках данной арифметики без перевода в иные системы счисления.

Программа демонстрирует корректную реализацию всех заданных функций. Программа создана по принципам ООП что удобно разделяет интерфейс и функционал класса, работающего с числами, и позволяет вносить изменения в программу легче, чем если бы она имела процедурную структуру.

Плюсы программы.

- Использование объектно-ориентированной парадигмы программирования, что позволяет удобно модифицировать программу.
- Все вычисления производятся напрямую, без использования заранее написанных константных таблиц.

Минусы программы.

- Результаты действий не сохраняются, что позволило бы проводить несколько действий подряд.
- Использование стандартных контейнеров библиотеки C++: вручную можно было бы создать более оптимизированные контейнеры.
- Неоптимизированный подход к выполнению арифметических действий: всё строится на правилах «+1» и «−1», что означает, что при масштабировании программы и включении возможности проводить действий с большим числом разрядов, действия будут проводиться медленнее, чем при использовании иного подхода для оптимальных вычислений.

Масштабирование. Используемая архитектура позволяет добавлять новые действия над числами данной алгебры (например, степени, НОК, НОД).

Для оптимизации арифметических действий можно сгенерировать матрицы переходов, содержащие не только правила «+1» и «−1», но и переносы разрядов. Это бы позволило не выполнять большое количество прибавлений единицы, а быстро находить ответ путём выбора нужных значений для каждого разряда.

Возможна интеграция с графическим интерфейсом или веб-интерфейсом. В будущем можно добавить возможность работы с несколькими действиями в одной строке.

Таким образом, реализованная программа является масштабируемой и обладает потенциалом развития. Её структура позволяет расширять функциональность и использовать в более крупных программных системах.

Список литературы

- [1] Павловская Т. А., Щюпак Ю. А. С++ Объектно-ориентированное программирование: Практикум. — СПб.: Питер, 2006 — 265 с.
- [2] 3. Хаггарти Р. Дискретная математика для программистов — Москва: ТЕХНОСФЕРА, 2012 - 400 с.
- [3] Секция "Телематика"(Электронный ресурс).
URL: <https://tema.spbstu.ru/dismath/>
(дата обращения: 24.11.2025).

Приложение 1. Файл Z8.h

```
1 #pragma once
2
3 #include <string>
4 #include <stdexcept>
5
6 class Z8Number {
7 public:
8     Z8Number();
9     explicit Z8Number(const std::string& s);
10    Z8Number(const Z8Number& other);
11
12    Z8Number operator+(const Z8Number& other) const;
13    Z8Number operator-(const Z8Number& other) const;
14    Z8Number operator*(const Z8Number& other) const;
15    std::string divide(const Z8Number& divisor, bool firstIsNeg = false) const;
16    Z8Number& operator=(const Z8Number& other);
17
18    bool operator==(const Z8Number& other) const;
19    std::string toString() const;
20
21 private:
22     std::string digits;
23     bool isNegative;
24
25     bool getNeg() const;
26     static char inc(char c);
27     static char dec(char c);
28     static std::string incNumber(const std::string& num);
29     static std::string decNumber(const std::string& num);
30     static std::string addNumbers(const std::string& x, const std::string& y);
```

```
31 static std::string subNumbers(const std::string& x, const std::string& y);
32 static std::string mulNumbers(const std::string& x, const std::string& y);
33 static std::string divNumbers(const std::string& x, const std::string& y, bool firstIsNeg =
34     false);
35 static bool greaterOrEqual(const std::string& x, const std::string& y);
36 friend bool greaterOrEqual(const Z8Number& x, const Z8Number& y);
37 friend bool greater(const Z8Number& x, const Z8Number& y);
38 static std::string normalize(const std::string& s);
39 static void validate(const std::string& s);
40 static bool isEqual(const std::string& a, const std::string& b);
41 friend void calculate(const Z8Number& a, const Z8Number& b, std::string op);
41 };
```

Листинг 1: Файл Z8.h

Приложение 2. Файл Z8.cpp

```
1 #include "z8.h"
2 #include <algorithm>
3 #include <stdexcept>
4 #include <iostream>
5
6 bool Z8Number::getNeg() const {
7     return isNegative;
8 }
9
10 char Z8Number::inc(char c) {
11     switch (c) {
12         case 'a': return 'b';
13         case 'b': return 'd';
14         case 'c': return 'e';
15         case 'd': return 'h';
16         case 'e': return 'g';
17         case 'f': return 'a';
18         case 'g': return 'f';
19         case 'h': return 'c';
20         default: throw std::invalid_argument("Invalid digit");
21     }
22 }
23
24 char Z8Number::dec(char c) {
25     switch (c) {
26         case 'a': return 'f';
27         case 'b': return 'a';
28         case 'c': return 'h';
29         case 'd': return 'b';
30         case 'e': return 'c';
31 }
```

```

31     case 'f': return 'g';
32     case 'g': return 'e';
33     case 'h': return 'd';
34     default: throw std::invalid_argument("Invalid digit");
35   }
36 }
37
38 // --- normalizing: getting rid of extra 'a' in the beginning ---
39 std::string Z8Number::normalize(const std::string& s) {
40   size_t i = s[0] == '-' ? 1 : 0;
41   while (i < s.length() - 1 && s[i] == 'a')
42     ++i;
43   return s.substr(i);
44 }
45
46 bool Z8Number::isEqual(const std::string& a, const std::string& b) {
47   return normalize(a) == normalize(b);
48 }
49
50 void Z8Number::validate(const std::string& s) {
51   if (s.empty()) throw std::invalid_argument("Empty number");
52   if (s.size() > 8 && s[0] != '-' || s.size() > 9) throw std::invalid_argument("More than 8
53   digits");
54   for (char c : s) {
55     if (c != 'a' && c != 'b' && c != 'c' && c != 'd' &&
56         c != 'e' && c != 'f' && c != 'g' && c != 'h' && c != '-')
57       throw std::invalid_argument("Invalid character");
58   }
59
60 Z8Number::Z8Number() : digits("a"), isNegative(false) {}
61 Z8Number::Z8Number(const std::string& s) {

```

```

62     validate(s);
63
64     digits = normalize(s);
65
66     isNegative = s[0] == '-' ? true : false;
67 }
68
69 Z8Number::Z8Number(const Z8Number& other) {
70
71     digits = other.digits;
72
73     isNegative = other.isNegative;
74 }
75
76 std::string Z8Number::incNumber(const std::string& num) {
77
78     std::string res = num;
79
80     int i = static_cast<int>(res.size()) - 1;
81
82     while (i >= 0) {
83
84         char old = res[i];
85
86         res[i] = inc(old);
87
88         if (old != 'f') { // no carry
89             break;
90         }
91
92         if (i == 0) { // carrying until no need to
93             res = 'b' + res;
94
95             if (res.size() > 8)
96                 throw std::overflow_error("Number exceeds 8 digits");
97
98             break;
99         }
100
101         --i;
102     }
103
104     if (normalize(res).size() > 8)
105         throw std::overflow_error("Number exceeds 8 digits");

```

```

94     return normalize(res);
95 }
96
97 std::string Z8Number::decNumber(const std::string& num) {
98     if (isEqual(num, "a"))
99         throw std::domain_error("Cannot decrement zero");
100    std::string res = num;
101    int i = static_cast<int>(res.size()) - 1;
102    while (i >= 0) {
103        char old = res[i];
104        res[i] = dec(old);
105        if (old != 'a') { // a -> f, so we need to take one from higher
106            break;
107        }
108        --i;
109    }
110    return normalize(res);
111 }
112
113 bool Z8Number::greaterOrEqual(const std::string& x, const std::string& y) {
114     if (isEqual(x, y)) return true;
115     if (isEqual(y, "a")) return true;
116     if (isEqual(x, "a")) return false;
117     try {
118         subNumbers(x, y);
119         return true;
120     }
121     catch (...) {
122         return false;
123     }
124 }
125

```

```

126 bool greater(const Z8Number& x, const Z8Number& y) {
127     if (x == y) return false;
128     if (y == Z8Number("a")) return true;
129     if (x == Z8Number("a")) return false;
130     try {
131         x - y;
132         return true;
133     }
134     catch (...) {
135         return false;
136     }
137 }
138
139 bool greaterOrEqual(const Z8Number& X, const Z8Number& Y) {
140     std::string x = X.digits;
141     std::string y = Y.digits;
142     return Z8Number::greaterOrEqual(x, y);
143 }
144
145 std::string Z8Number::addNumbers(const std::string& x, const std::string& y) {
146     if (isEqual(y, "a")) return x;
147     std::string res = x;
148     std::string counter = "a";
149     while (!isEqual(counter, y)) {
150         res = incNumber(res);
151         counter = incNumber(counter);
152         if (normalize(res).size() > 8)
153             throw std::overflow_error("Number exceeds 8 digits");
154     }
155     return res;
156 }
157

```

```

158 std::string Z8Number::subNumbers(const std::string& x, const std::string& y) {
159     if (isEqual(y, "a")) return x;
160     if (isEqual(x, y)) return "a";
161     std::string res = x;
162     std::string counter = "a";
163     while (!isEqual(counter, y)) {
164         res = decNumber(res);
165         counter = incNumber(counter);
166     }
167     return res;
168 }
169
170 std::string Z8Number::mulNumbers(const std::string& x, const std::string& y) {
171     if (isEqual(x, "a") || isEqual(y, "a")) return "a";
172     std::string res = x;
173     std::string counter = "b";
174     if (isEqual(y, "ba")) res = res + "a";
175     else if (isEqual(y, "baa")) res = res + "aa";
176     else if (isEqual(y, "baaa")) res = res + "aaa";
177     else if (isEqual(y, "baaaa")) res = res + "aaaa";
178     else if (isEqual(y, "baaaaa")) res = res + "aaaaa";
179     else if (isEqual(y, "baaaaaa")) res = res + "aaaaaa";
180     else if (isEqual(y, "baaaaaaa")) res = res + "aaaaaaa";
181     else {
182         while (!isEqual(counter, y)) {
183             res = addNumbers(res, x);
184             counter = incNumber(counter);
185         }
186     }
187     if (normalize(res).size() > 8)
188         throw std::overflow_error("Number exceeds 8 digits");
189     return res;

```

```

190 }
191
192 std::string Z8Number::divNumbers(const std::string& x, const std::string& y, bool firstIsNeg)
193 {
194     if (isEqual(y, "a")) {
195         if (isEqual(x, "a"))
196             return "-fffffff; fffffff";
197         else
198             return "empty set";
199     }
200     if (isEqual(x, "a")) {
201         return "a";
202     }
203     std::string quotient = "a";
204     std::string remainder = x;
205
206     while (greaterOrEqual(remainder, y)) {
207         remainder = subNumbers(remainder, y);
208         quotient = incNumber(quotient);
209     }
210
211     if (firstIsNeg && !isEqual(remainder, "a")) {
212         quotient = incNumber(quotient);
213         remainder = subNumbers(y, remainder);
214     }
215
216     if (isEqual(remainder, "a")) {
217         return quotient;
218     }
219     else {
220         return quotient + "(" + remainder + ")";

```

```

221     }
222 }
223
224 std::string Z8Number::divide(const Z8Number& divisor, bool firstIsNeg) const {
225     return divNumbers(digits, divisor.digits, firstIsNeg);
226 }
227
228 Z8Number Z8Number::operator+(const Z8Number& o) const {
229     return Z8Number(addNumbers(digits, o.digits));
230 }
231
232 Z8Number Z8Number::operator-(const Z8Number& o) const {
233     return Z8Number(subNumbers(digits, o.digits));
234 }
235
236 Z8Number Z8Number::operator*(const Z8Number& o) const {
237     return Z8Number(mulNumbers(digits, o.digits));
238 }
239
240 Z8Number& Z8Number::operator=(const Z8Number& other) {
241     digits = other.digits;
242     isNegative = other.isNegative;
243     return *this;
244 }
245
246 std::string Z8Number::toString() const {
247     return digits;
248 }
249
250 void calculate(const Z8Number& a, const Z8Number& b, std::string op) {
251     std::string result;
252     if (!(a.isNegative) && !(b.isNegative))) {

```

```

253     try {
254         if (op == "+") {
255             result = (a + b).toString();
256         }
257         else if (op == "-") {
258             try {
259                 result = (a - b).toString();
260             }
261             catch (...) {
262                 result = "- " + (b - a).toString();
263             }
264         }
265         else if (op == "*") {
266             if (greater(a, b)) result = (a * b).toString();
267             else result = (b * a).toString();
268         }
269         else if (op == "/") {
270             result = a.divide(b);
271         }
272         else {
273             std::cerr << "Unknown operator\n";
274         }
275     }
276     catch (const std::exception& e) {
277         std::cerr << "Error: " << e.what() << "\n";
278     }
279 }
280 else if ((a.isNegative) && (!(b.isNegative)) || (!(a.isNegative)) && (b.isNegative)) {
281     Z8Number neg, pos;
282     if ((a.isNegative) && (!(b.isNegative))) {
283         neg = a;
284         pos = b;

```

```

285     }
286     else {
287         neg = b;
288         pos = a;
289     }
290     try {
291         if (op == "+") {
292             if (greater(neg, pos)) result = "-" + (neg - pos).toString();
293             else result = (pos - neg).toString();
294         }
295         else if (op == "-") {
296             result = (neg + pos).toString();
297         }
298         else if (op == "*") {
299             if (a == Z8Number("a") || a == Z8Number("-a") || b == Z8Number("a") ||
300                 b == Z8Number("-a")) result = "a";
301             else result = "-" + (a * b).toString();
302         }
303         else if (op == "/") {
304             if (a == Z8Number("a")) result = a.divide(b);
305             else if (a.isNegative) result = "-" + a.divide(b, true);
306             else result = "-" + a.divide(b);
307         }
308         std::cerr << "Unknown operator\n";
309     }
310 }
311 catch (const std::exception& e) {
312     std::cerr << "Error: " << e.what() << "\n";
313 }
314 }
315 else {

```

```

316     try {
317         if (op == "+") {
318             if ((a == Z8Number("a") || a == Z8Number("-a")) && (b == Z8Number("a") || b == Z8Number("-a"))) result = "a";
319             else result = "-+" + (a + b).toString();
320         }
321         else if (op == "-") {
322             if (a == b) result = "a";
323             else if ((a == Z8Number("a") || a == Z8Number("-a")) && (b == Z8Number("a") || b == Z8Number("-a"))) result = "a";
324             else {
325                 try {
326                     result = "-+" + (a - b).toString();
327                 }
328                 catch (...) {
329                     result = (b - a).toString();
330                 }
331             }
332         }
333         else if (op == "*") {
334             result = (a * b).toString();
335         }
336         else if (op == "/") {
337             result = a.divide(b);
338         }
339         else {
340             std::cerr << "Unknown operator\n";
341         }
342     }
343     catch (const std::exception& e) {
344         std::cerr << "Error: " << e.what() << "\n";
345     }

```

```
346 }
347 if (result == "-empty set") result = "empty set";
348 std::cout << result << std::endl;
349 }
```

Листинг 2: Файл Z8.cpp

Приложение 3. Файл main.cpp

```
1 #include <iostream>
2 #include <string>
3 #include "z8.h"
4
5 int main() {
6     /*std::cout << "==== DEBUG ====" << std::endl;
7     std::cout << "a + b = " << (Z8Number("a") + Z8Number("b")).toString() << "\n";
8     std::cout << "f + b = " << (Z8Number("f") + Z8Number("b")).toString() << "\n";
9     std::cout << "a * d = " << (Z8Number("a") * Z8Number("d")).toString() << "\n";
10    std::cout << "ff * f = " << (Z8Number("ff") * Z8Number("f")).toString() << "\n";
11    std::cout << "aab * c = " << (Z8Number("aab") * Z8Number("c")).toString() <<
12        "\n";
13    std::cout << "f - b = " << (Z8Number("f") - Z8Number("b")).toString() << "\n";
14    std::cout << "ba - d = " << (Z8Number("ba") - Z8Number("d")).toString() << "\n";
15    std::cout << "e + e = " << (Z8Number("e") + Z8Number("e")).toString() << "\n";
16    std::cout << "ba / e = " << Z8Number("ba").divide(Z8Number("e")) << "\n";
17    std::cout << "-ba + b = "; calculate(Z8Number("-ba"), Z8Number("b"), "+");
18    std::cout << "-b + f = "; calculate(Z8Number("-b"), Z8Number("f"), "+");
19    std::cout << "-ba - ff = "; calculate(Z8Number("-ba"), Z8Number("-ff"), "-");
20    std::cout << "b - d = "; calculate(Z8Number("b"), Z8Number("d"), "-");
21    std::cout << "-a + a = "; calculate(Z8Number("-a"), Z8Number("a"), "+");
22    std::cout << "ba * baa = "; calculate(Z8Number("ba"), Z8Number("baa"), "*");
23    std::cout << "-a - -a ="; calculate(Z8Number("-a"), Z8Number("-a"), "-");
24    std::cout << "-a + -a ="; calculate(Z8Number("-a"), Z8Number("-a"), "+");
25    std::cout << "-a * a ="; calculate(Z8Number("-a"), Z8Number("a"), "*");
26    std::cout << "-a * baa ="; calculate(Z8Number("-a"), Z8Number("baa"), "*");
27    std::cout << "Hello!" << std::endl;
28    std::string s1, s2, op;
29    while (true) {
```

```
30     std::cout << "Enter the problem (format: <num> <operation> <num>; nums from  
31         -ffffffff to ffffffff):\n>> ";  
32     std::cin >> s1;  
33     std::cin >> op;  
34     std::cin >> s2;  
35  
36     try {  
37         Z8Number a(s1), b(s2);  
38         calculate(a, b, op);  
39     }  
40     catch (...) {  
41         std::cerr << "Error: incorrect input (invalid characters)" << std::endl;  
42     }  
43 }
```

Листинг 3: Файл main.cpp