

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 дисциплине

Дискретная математика

Курсовая работа

«Калькулятор конечной арифметики Z_8 »

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 Исходная структура Z_8	5
2.2 Арифметические операции	5
2.3 Ограничения	8
3 Особенности реализации	10
3.1 Реализация алгебраической логики	10
3.2 Реализация консольного меню	21
4 Результаты работы программы	22
Заключение	25
Список литературы	27
Приложение 1. Файл Z8.h	28
Приложение 2. Файл Z8.cpp	30
Приложение 3. Файл main.cpp	41

Введение

Актуальность и цель работы

Использование кода Грея позволяет минимизировать количество ошибок при переходе от одного состояния к другому, что применяется, например, в кодировании данных. Мультимножества позволяют описывать системы, где элемент может встречаться несколько раз — это востребовано в базах данных, статистике и других областях.

Таким образом, разработка программного инструмента, который демонстрирует генерацию кода Грея и операции над мультимножествами, способствует формированию навыков работы с дискретными структурами данных и алгоритмами, а также помогает на практике закрепить фундаментальные понятия теории множеств.

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 «b», а также выполняется свойство: $(\forall x) * a = a$.

Цель данной работы — реализовать программу, которая:

- поддерживает сложение для данной арифметики,
- поддерживает разность для данной арифметики,
- поддерживает умножение для данной арифметики,
- поддерживает деление для данной арифметики,
- поддерживает мультиразрядность (возможна операция $bbb + ccc$),
- поддерживает отрицательные числа,
- выполняет вычисления только в рамках данной арифметики без перевода в иные системы счисления.

2 Математическое описание

2.1 Конечная арифметика Z_8

Рассмотрим нашу конечную арифметику

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

в которой задана операция « $+1$ », 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

Это правило задаёт циклическую аддитивную структуру порядка 8. Элемент a является **аддитивной единицей**:

$$\forall x \in Z_8, \quad x + a = x.$$

Элемент b задан как **мультипликативная единица**:

$$\forall x \in Z_8, \quad x \cdot b = x.$$

Выполняется свойство поглощения:

$$\forall x \in Z_8, \quad x \cdot a = a.$$

В этой алгебре выполняются следующие свойства: коммутативность, ассоциативность и дистрибутивность.

Эти свойства позволяют рассматривать структуру $\langle Z_8; +, \cdot \rangle$ как **коммутативное кольцо с единицей**.

2.2 Арифметические операции

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

Сложение

Сложение выполняется по следующей логике.

1. Пусть надо сложить числа A и B .
2. Выбирается наибольшее из этих двух чисел (можно, т.к. данная алгебра коммутативна). Пусть наибольшее в данном случае A .
3. К A прибавляется элемент b (мультипликативная единица) B раз. Это, в свою очередь, производится по следующей логике.

- (a) Если разряд, к которому прибавлялось b , не был равен f до прибавления, измененным остается только текущий разряд.
- (b) 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 во втором пункте обоснован оптимизацией вычислений, т.к. если мы складываем, например, b и $ffffffg$, намного быстрее прибавить один раз b к $ffffffg$, чем наоборот.

Умножение

Умножение выполняется по следующей логике.

1. Пусть надо умножить число A на B .
2. Выбирается наибольшее из этих двух чисел (коммутативность). Пусть наибольшее в данном случае A .
3. К A прибавляется A $B-1$ раз, что реализовано через правило "+1".

Выбор наибольшего из двух чисел во втором пункте обоснован оптимизацией вычислений, т.к. если мы умножим, например, `baa` и `baaa`, намного быстрее прибавлять к себе `baaa` `baa` раз, чем наоборот.

Вычитание

Вдобавок 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 операция 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 операции с ними

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

Операции с отрицательными числами проводятся по следующей логике:

Если отрицательно одно число:

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

Если оба числа отрицательны:

1. Пусть модули двух отрицательных чисел A и B соответственно.
2. При сложении складываем модули и приписываем минус.
3. При вычитании берем модуль большего, вычитаем из него модуль меньшего; если $A > B$, приписываем минус, в противном случае нет.
4. При умножении совершаем умножение модулей.
5. При делении совершается деление модулей.

2.3 Ограничения

В данной алгебре присутствуют следующие ограничения:

- нельзя делить на ноль;
- нельзя проводить операции над числами, превышающими 8 разрядов;
- результат операции не должен превышать 8 разрядов.

Выход за эти ограничения ведет к неопределенному выходу, т.е. результат таких операций неизвестен.

3 Особенности реализации

3.1 Реализация алгебраической логики

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

В рамках данного описания числами будут называться элементы структуры Z_8 (`a`, `b`, ..., `f`).

- **Метод** `getNeg`

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

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

Назначение: возвращает булево 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 операции "+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 операции
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 'a'.

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

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] == 'a')  
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
        than 8 digits");
4     for (char c : s) {
5         if (c != 'a' && c != 'b' && c != 'c' && c != 'd' &&
6             c != 'e' && c != 'f' && c != 'g' && c != 'h' && c != '-')
7             throw std::invalid_argument("Invalid character");
8     }
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 }
```


12 }

- **Метод** `subNumbers`

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

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 + "aaaaaaa";
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
    firstIsNeg) {
2     if (isEqual(y, "a")) {
3         throw std::domain_error("Division by zero");
4     }
5     if (isEqual(x, "a")) {
6         return "a";

```

```

7      }
8
9      std::string quotient = "a";
10     std::string remainder = x;
11
12     while (greaterOrEqual(remainder, y)) {
13         remainder = subNumbers(remainder, y);
14         quotient = incNumber(quotient);
15     }
16
17     if (firstIsNeg && !isEqual(remainder, "a")) {
18         quotient = incNumber(quotient);
19         remainder = subNumbers(y, remainder);
20     }
21
22     if (isEqual(remainder, "a")) {
23         return quotient;
24     }
25     else {
26         return quotient + "(" + remainder + ")";
27     }
28 }

```

- **Метод** `operator +, -, *`

Вход: `Z8Number num1, Z8Number num2` - числа для проведения арифметической операции.

Выход: `Z8Number result` - результат выполнения арифм. операции.

Назначение: выполнение операции 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 операция над ними.

Выход: `Z8Number result` - результат выполнения арифметической операции над числами.

Назначение: выполнение арифметической операции 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            ...

```

3.2 Реализация консольного меню

Консольное меню реализовано в файле `main.py`. 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 -
        fffffff to fffffff):\n">> ";
5     std::cin >> s1;
6     std::cin >> op;
7     std::cin >> s2;
8
9     try {
10         Z8Number a(s1), b(s2);
11         calculate(a, b, op);
12     }
13     catch (...) {
14         std::cerr << "Error: incorrect input" << std::endl;
15     }
16 }
```

4 Результаты работы программы

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

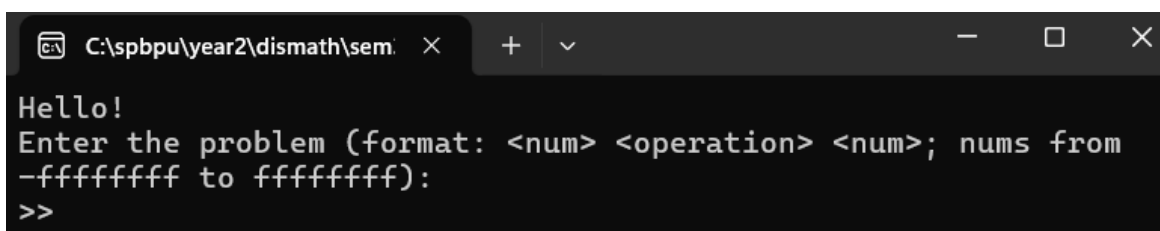
Меню программы

После запуска программы пользователю предлагается ввести пример, который надо решить, в формате <число> <операция> <число>. В случае некорректного ввода (неверный формат, превышение лимитов, некорректные

значения чисел) или ошибки в результате вычислений (деление на ноль, превышение лимитов) программа сообщит об этом пользователю.

Пример работы

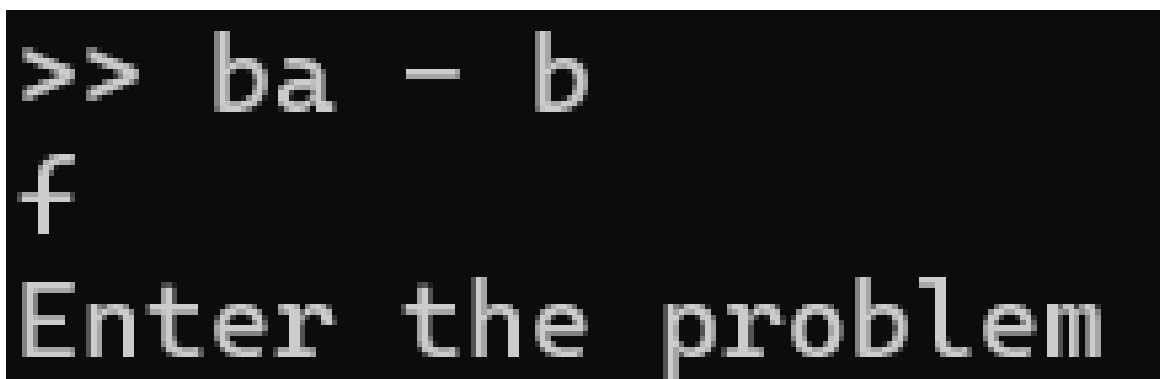
При запуске программы высвечивается приветственное окно и предложение ввести пример. (рис. 1).



```
C:\spbpu\year2\dismath\sem1 > .\program.exe
Hello!
Enter the problem (format: <num> <operation> <num>; nums from
-ffffffff to fffffffff):
>>
```

Рис. 1: Приветственное сообщение при запуске программы.

Когда пользователь введет пример и нажмет Enter, программа выведет результат вычислений (рис. 2).



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

Рис. 2: Вывод решения примера.

Если пользователь сделает некорректный ввод, программа об этом сообщит и предложит ввести пример снова (рис. 3).

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

Рис. 3: Обработка некорректного пользовательского ввода.

Если результат операции превышает лимиты, программа так же сообщает об этом (рис. 4).

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

Рис. 4: Обработка попытки проведения операций над несуществующими мультимножествами.

Результаты подтвердили, что программа реализована корректно.

Заключение

В ходе лабораторной работы была разработана программа, позволяющая:

- 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 мультиразрядностью,
- работать с отрицательными числами,
- обрабатывать некорректный ввод пользователя и ограничения в операциях,
- 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.

Минусы программы.

- Результаты операций не сохраняются, что позволило бы проводить несколько операций подряд.
- Использование стандартных контейнеров библиотеки: вручную можно было бы создать более оптимизированные контейнеры.
- Неоптимизированный подход к выполнению арифметических операций: всё строится на правиле $+1$, что означает, что при масштабировании программы и включении возможности проводить операции с большим числом разрядов, операции будут проводиться медленнее, чем при использовании большего числа матриц переходов для оптимальных вычислений.

Масштабирование. Используемая архитектура позволяет добавлять новые операции над числами данной алгебры (например, степени, НОК, НОД). Возможна интеграция с графическим интерфейсом или веб-интерфейсом. В будущем можно добавить возможность работы с несколькими операциями 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] З. Хаггарт 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 =
        false);
34     static bool greaterOrEqual(const std::string& x, const std::string& y);
35     friend bool greaterOrEqual(const Z8Number& x, const Z8Number& y);
36     friend bool greater(const Z8Number& x, const Z8Number& y);
37     static std::string normalize(const std::string& s);
38     static void validate(const std::string& s);
39     static bool isEqual(const std::string& a, const std::string& b);
40     friend void calculate(const Z8Number& a, const Z8Number& b, std::string op);
41 };
42
43 //class OverflowError : public std::overflow_error {
44 //public:
45 // OverflowError() : std::overflow_error("Number exceeds 8 digits") {}
46 //};

```

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     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     digits = normalize(s);
64     isNegative = s[0] == '-' ? true : false;
65 }
66 Z8Number::Z8Number(const Z8Number& other) {
67     digits = other.digits;
68     isNegative = other.isNegative;
69 }
70
71 bool Z8Number::operator==(const Z8Number& other) const {
72     return isEqual(digits, other.digits);
73 }
74
75 std::string Z8Number::incNumber(const std::string& num) {
76     std::string res = num;
77     int i = static_cast<int>(res.size()) - 1;
78     while (i >= 0) {
79         char old = res[i];
80         res[i] = inc(old);
81         if (old != 'f') { // no carry
82             break;
83         }
84         if (i == 0) { // carrying until no need to
85             res = 'b' + res;
86             if (res.size() > 8)
87                 throw std::overflow_error("Number exceeds 8 digits");
88             break;
89         }
90         --i;
91     }
92     if (normalize(res).size() > 8)
93         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 + "aaaaaaa";
180     else if (isEqual(y, "baaaaaaa")) res = res + "aaaaaaaa";
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         throw std::domain_error("Division by zero");
196     }
197     if (isEqual(x, "a")) {
198         return "a";
199     }
200
201     std::string quotient = "a";
202     std::string remainder = x;
203
204     while (greaterOrEqual(remainder, y)) {
205         remainder = subNumbers(remainder, y);
206         quotient = incNumber(quotient);
207     }
208
209     if (firstIsNeg && !isEqual(remainder, "a")) {
210         quotient = incNumber(quotient);
211         remainder = subNumbers(y, remainder);
212     }
213
214     if (isEqual(remainder, "a")) {
215         return quotient;
216     }
217     else {
218         return quotient + "(" + remainder + ")";
219     }
220 }

```

```

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

```

```

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

```

```

284     }
285     else {
286         neg = b;
287         pos = a;
288     }
289     try {
290         if (op == "+") {
291             if (greater(neg, pos)) result = "-" + (neg - pos).toString();
292             else result = (pos - neg).toString();
293         }
294         else if (op == "-") {
295             result = (neg + pos).toString();
296         }
297         else if (op == "*") {
298             if (a == Z8Number("a") || a == Z8Number("-a") || b == Z8Number("a") ||
299                 b == Z8Number("-a")) result = "a";
300             else result = "-" + (a * b).toString();
301         }
302         else if (op == "/") {
303             if (a == Z8Number("a")) result = a.divide(b);
304             else if (a.isNegative) result = "-" + a.divide(b, true);
305             else result = "-" + a.divide(b);
306         }
307         else {
308             std::cerr << "Unknown operator\n";
309         }
310     } catch (const std::exception& e) {
311         std::cerr << "Error: " << e.what() << "\n";
312     }
313 }
314 else {

```

```

315     try {
316         if (op == "+") {
317             if ((a == Z8Number("a") || a == Z8Number("-a")) && (b == Z8Number("
318                 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 ==
324                 Z8Number("a") || b == Z8Number("-a"))) result = "a";
325             else {
326                 try {
327                     result = "-" + (a - b).toString();
328                 }
329                 catch (...) {
330                     result = (b - a).toString();
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         }

```



```
345     }  
346     std::cout << result << std::endl;  
347 }
```

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 << "aaab * c = " << (Z8Number("aaab") + Z8Number("c")).toString() <<
        "\n";
12    std::cout << "f - b = " << (Z8Number("f") - Z8Number("b")).toString() << "\n";
13    std::cout << "ba - d = " << (Z8Number("ba") - Z8Number("d")).toString() << "\n";
14    std::cout << "e + e = " << (Z8Number("e") + Z8Number("e")).toString() << "\n";
15    std::cout << "ba / e = " << Z8Number("ba").divide(Z8Number("e")) << "\n";
16    std::cout << "-ba + b = "; calculate(Z8Number("-ba"), Z8Number("b"), "+");
17    std::cout << "-b + f = "; calculate(Z8Number("-b"), Z8Number("f"), "+");
18    std::cout << "-ba - ff = "; calculate(Z8Number("-ba"), Z8Number("-ff"), "-");
19    std::cout << "b - d = "; calculate(Z8Number("b"), Z8Number("d"), "-");
20    std::cout << "-a + a = "; calculate(Z8Number("-a"), Z8Number("a"), "+");
21    std::cout << "ba * baa = "; calculate(Z8Number("ba"), Z8Number("baa"), "*");
22    std::cout << "-a - -a = "; calculate(Z8Number("-a"), Z8Number("-a"), "-");
23    std::cout << "-a + -a = "; calculate(Z8Number("-a"), Z8Number("-a"), "+");
24    std::cout << "-a * a = "; calculate(Z8Number("-a"), Z8Number("a"), "*");
25    std::cout << "-a * baa = "; calculate(Z8Number("-a"), Z8Number("baa"), "*");*/
26
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
        -ffffff to fffffff):\n>> ";
31     std::cin >> s1;
32     std::cin >> op;
33     std::cin >> s2;
34
35     try {
36         Z8Number a(s1), b(s2);
37         calculate(a, b, op);
38     }
39     catch (...) {
40         std::cerr << "Error: incorrect input (invalid characters)" << std::endl;
41     }
42 }
43 }

```

Листинг 3: Файл main.cpp