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

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 $< Z_i; +, * >$ (8 разрядов) для четырех действий (+, -, *, /) на основе "малой"конечной арифметики, где задано правило «+1» и выполняются свойства коммутативности (+,*), ассоциативности (+, *), дистрибутивности * относительно +, заданы аддитивная единица «a» 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,
- подддерживает 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 Исходная структура — «малая» конечная арифметика

Рассмотрим конечное множество

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

на котором 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 умножем, 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 приписываеим 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 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 операции "+1"над текущим.

  Код:

```
1 char Z8Number::inc(char c) {
2     switch (c) {
3         case 'a': return 'b';
```

```
        case 'b': return 'd';
        case 'c': return 'e';
        case 'd': return 'h';
        case 'e': return 'g';
        case 'f': return 'a';
        case 'g': return 'f';
        case 'h': return 'c';
        default: throw std::invalid_argument("Invalid digit");
    }
}
```

- **Метод** `dec`

  Вход: `char c` - текущее число.

  Выход: `char c` - предыдущее число.

  Назначение: возвращает число, получающееся в результате операции 1"над текущим.

  Код:

```
char Z8Number::dec(char c) {
    switch (c) {
        case 'a': return 'f';
        case 'b': return 'a';
        case 'c': return 'h';
        case 'd': return 'b';
        case 'e': return 'c';
        case 'f': return 'g';
        case 'g': return 'e';
        case 'h': return 'd';
        default: throw std::invalid_argument("Invalid digit");
    }
}
```

- **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:

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

- **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:

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

- **Метод** `validate`

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

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

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

  Код:

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

- **Метод** `operator==`

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

  Выход: `bool n` - результат сравнения двух чисел.

  Назначение: сравнивает два числа по модулю.

  Код:

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

- **Метод** `incNumber`

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

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

  Назначение: увеличивает число на единицу, учитывая переносы.

  Код:

```
std::string Z8Number::incNumber(const std::string& num) {
    std::string res = num;
    int i = static_cast<int>(res.size()) - 1;
    while (i >= 0) {
```

```
        char old = res[i];
        res[i] = inc(old);
        if (old != 'f') { // no carry
            break;
        }
        if (i == 0) { // carrying until no need to
            res = 'b' + res;
            if (res.size() > 8)
            throw OverflowError();
            break;
        }
        --i;
    }
    return normalize(res);
}
```

- **Метод** `decNumber`

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

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

  Назначение: уменьшает число на единицу, учитывая переносы.

  Код:

```
std::string Z8Number::decNumber(const std::string& num) {
    if (isEqual(num, "a"))
    throw std::domain_error("Cannot decrement zero");
    std::string res = num;
    int i = static_cast<int>(res.size()) - 1;
    while (i >= 0) {
        char old = res[i];
        res[i] = dec(old);
        if (old != 'a') { // a -> f, so we need to take one from higher
            break;
```

```
        }
        --i;
    }
    return normalize(res);
}
```

- **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:

```
bool Z8Number::greaterOrEqual(const std::string& x, const std::string& y) {
    if (isEqual(x, y)) return true;
    if (isEqual(y, "a")) return true;
    if (isEqual(x, "a")) return false;
    try {
        subNumbers(x, y);
        return true;
    }
    catch (...) {
        return false;
    }
}
```

- **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:

```cpp
bool greater(const Z8Number& x, const Z8Number& y) {
    if (x == y) return false;
    if (y == Z8Number("a")) return true;
    if (x == Z8Number("a")) return false;
    try {
        x - y;
        return true;
    }
    catch (...) {
        return false;
    }
}
```

- **Метод** `addNumbers`

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

  Выход: `std::string result` - результат суммирования.

  Назначение: выполненяет сложение двух чисел.

  Код:

```cpp
std::string Z8Number::addNumbers(const std::string& x, const std::string& y) {
    if (isEqual(y, "a")) return x;
    std::string res = x;
    std::string counter = "a";
    while (!isEqual(counter, y)) {
        res = incNumber(res);
        counter = incNumber(counter);
        if (normalize(res).size() > 8)
        throw std::overflow_error("Number exceeds 8 digits");
    }
    return res;
}
```

- **Метод** `subNumbers`

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

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

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

  Код:

```
std::string Z8Number::subNumbers(const std::string& x, const std::string& y) {
    if (isEqual(y, "a")) return x;
    if (isEqual(x, y)) return "a";
    std::string res = x;
    std::string counter = "a";
    while (!isEqual(counter, y)) {
        res = decNumber(res);
        counter = incNumber(counter);
    }
    return res;
}
```

- **Метод** `mulNumbers`

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

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

  Назначение: выполненяет умножение двух чисел.

  Код:

```
std::string Z8Number::mulNumbers(const std::string& x, const std::string& y) {
    if (isEqual(x, "a") || isEqual(y, "a")) return "a";
    std::string res = x;
    std::string counter = "b";
    if (isEqual(y, "ba")) res = res + "a";
    else if (isEqual(y, "baa")) res = res + "aa";
```

```
    else if (isEqual(y, "baaa")) res = res + "aaa";
    else if (isEqual(y, "baaaa")) res = res + "aaaa";
    else if (isEqual(y, "baaaaa")) res = res + "aaaaa";
    else if (isEqual(y, "baaaaaa")) res = res + "aaaaaaa";
    else if (isEqual(y, "baaaaaaa")) res = res + "aaaaaaa";
    else {
        while (!isEqual(counter, y)) {
            res = addNumbers(res, x);
            counter = incNumber(counter);
        }
    }
    if (normalize(res).size() > 8)
    throw std::overflow_error("Number exceeds 8 digits");
    return res;
}
```

- **Метод** `divNumbers`, `divide`

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

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

  Назначение: выполненяет деление одного числа на другое.

  Код:

```
std::string Z8Number::divNumbers(const std::string& x, const std::string& y, bool
    oneIsNeg) {
    if (isEqual(y, "a")) {
        throw std::domain_error("Division by zero");
    }
    if (isEqual(x, "a")) {
        return "a";
    }

    std::string quotient = "a";
```

```
    std::string remainder = x;

    while (greaterOrEqual(remainder, y)) {
        remainder = subNumbers(remainder, y);
        quotient = incNumber(quotient);
    }

    if (oneIsNeg && !isEqual(remainder, "a")) {
        quotient = incNumber(quotient);
        remainder = subNumbers(y, remainder);
    }

    if (isEqual(remainder, "a")) {
        return quotient;
    }
    else {
        return quotient + "(" + remainder + ")";
    }
}
```

- **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:

```
Z8Number Z8Number::operator+(const Z8Number& o) const {
    return Z8Number(addNumbers(digits, o.digits));
}
Z8Number Z8Number::operator-(const Z8Number& o) const {
```

```
    return Z8Number(subNumbers(digits, o.digits));
}
Z8Number Z8Number::operator*(const Z8Number& o) const {
    return Z8Number(mulNumbers(digits, o.digits));
}
```

- **Метод** `operator =`

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

  Выход: `Z8Number result` - обновленное текущее число.

  Назначение: присваивает значение другого числа текущему.

  Код:

```
Z8Number& Z8Number::operator=(const Z8Number& other) {
    digits = other.digits;
    isNegative = other.isNegative;
    return *this;
}
```

- **Метод** `toString`

  Вход: `Z8Number num` - текущее число.

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

  Назначение: переводит число в строку.

  Код:

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

- **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:

```
void calculate(const Z8Number& a, const Z8Number& b, std::string op) {
    std::string result;
    if (!(a.isNegative) && (!(b.isNegative))) {
        try {
            if (op == "+") {
                result = (a + b).toString();
            }
            else if (op == "-") {
                try {
                    result = (a - b).toString();
                }
                catch (...) {
                    result = "-" + (b - a).toString();
                }
            }
...
```

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:

```
std::cout << "Hello!" << std::endl;
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 некорретного 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).

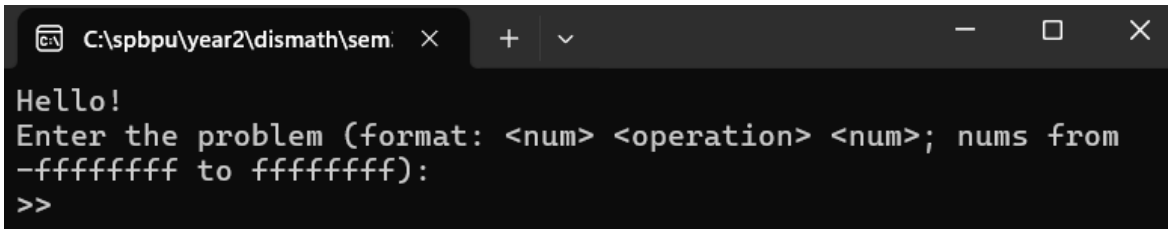

Рис. 1: Приветственное сообщение при запуске программы.

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

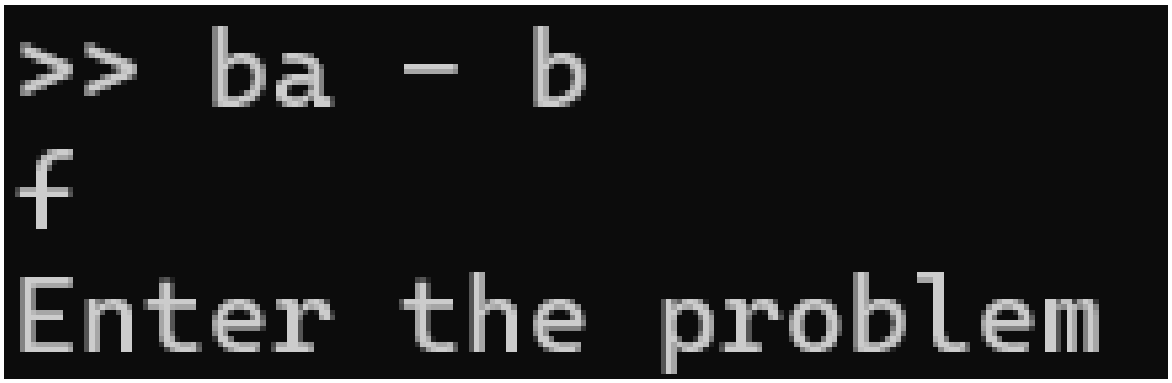

Рис. 2: Вывод решения примера.

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

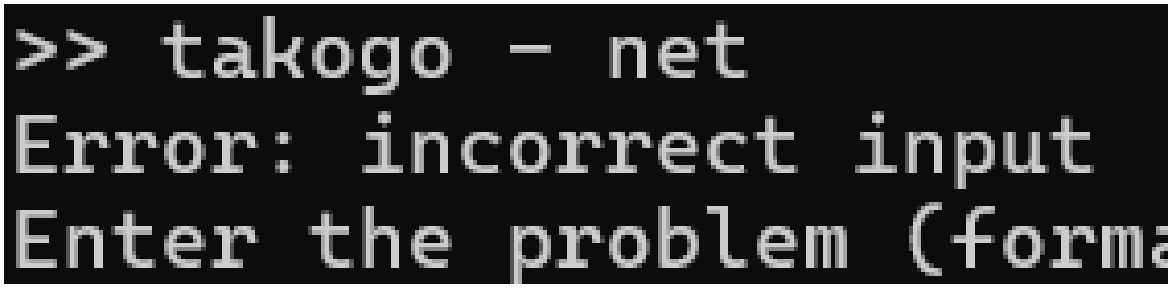

Рис. 3: Обработка некорретного пользовательского ввода.

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

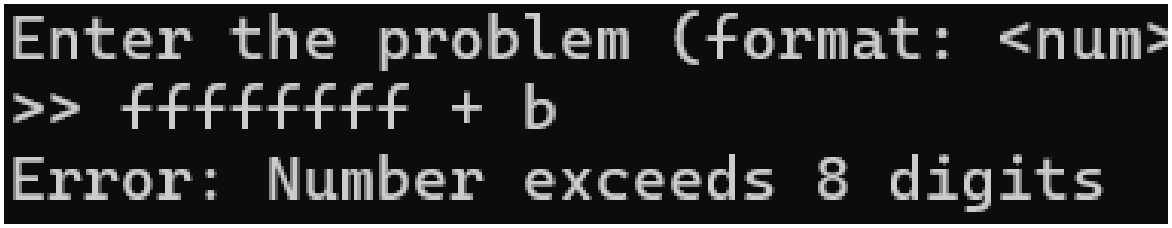

Рис. 4: Обработка попытки проведения операций над несуществующими мультимножествами.

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. C++ 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.

```
#pragma once

#include <string>
#include <stdexcept>

class Z8Number {
public:
    Z8Number();
    explicit Z8Number(const std::string& s);
    Z8Number(const Z8Number& other);

    Z8Number operator+(const Z8Number& other) const;
    Z8Number operator-(const Z8Number& other) const;
    Z8Number operator*(const Z8Number& other) const;
    std::string divide(const Z8Number& divisor) const;
    Z8Number& operator=(const Z8Number& other);

    bool operator==(const Z8Number& other) const;
    std::string toString() const;

private:
    std::string digits;
    bool isNegative;

    bool getNeg() const;
    static char inc(char c);
    static char dec(char c);
    static std::string incNumber(const std::string& num);
    static std::string decNumber(const std::string& num);
    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);
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.

```
#include "z8.h"
#include <algorithm>
#include <stdexcept>
#include <iostream>

bool Z8Number::getNeg() const {
    return isNegative;
}

char Z8Number::inc(char c) {
    switch (c) {
    case 'a': return 'b';
    case 'b': return 'd';
    case 'c': return 'e';
    case 'd': return 'h';
    case 'e': return 'g';
    case 'f': return 'a';
    case 'g': return 'f';
    case 'h': return 'c';
    default: throw std::invalid_argument("Invalid digit");
    }
}

char Z8Number::dec(char c) {
    switch (c) {
    case 'a': return 'f';
    case 'b': return 'a';
    case 'c': return 'h';
    case 'd': return 'b';
    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
            digits");
53          for (char c : s) {
54              if (c != 'a' && c != 'b' && c != 'c' && c != 'd' &&
55                  c != 'e' && c != 'f' && c != 'g' && c != 'h' && c != '-')
56                  throw std::invalid_argument("Invalid character");
57          }
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 OverflowError();
88              break;
89          }
90          --i;
91      }
92      return normalize(res);
93  }
```

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

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

```
158     std::string counter = "a";
159     while (!isEqual(counter, y)) {
160         res = decNumber(res);
161         counter = incNumber(counter);
162     }
163     return res;
164 }
165 
166 std::string Z8Number::mulNumbers(const std::string& x, const std::string& y) {
167     if (isEqual(x, "a") || isEqual(y, "a")) return "a";
168     std::string res = x;
169     std::string counter = "b";
170     while (!isEqual(counter, y)) {
171         res = addNumbers(res, x);
172         counter = incNumber(counter);
173     }
174     return res;
175 }
176 
177 std::string Z8Number::divNumbers(const std::string& x, const std::string& y) {
178     if (isEqual(y, "a")) {
179         throw std::domain_error("Division by zero");
180     }
181     if (isEqual(x, "a")) {
182         return "a";
183     }
184 
185     std::string quotient = "a";
186     std::string remainder = x;
187 
188     while (greaterOrEqual(remainder, y)) {
189         remainder = subNumbers(remainder, y);
```

```
190             quotient = incNumber(quotient);
191         }
192 
193         if (isEqual(remainder, "a")) {
194             return quotient;
195         }
196         else {
197             return quotient + "." + remainder;
198         }
199 }
200 
201 std::string Z8Number::divide(const Z8Number& divisor) const {
202         return divNumbers(digits, divisor.digits);
203 }
204 
205 Z8Number Z8Number::operator+(const Z8Number& o) const {
206         return Z8Number(addNumbers(digits, o.digits));
207 }
208 
209 Z8Number Z8Number::operator-(const Z8Number& o) const {
210         return Z8Number(subNumbers(digits, o.digits));
211 }
212 
213 Z8Number Z8Number::operator*(const Z8Number& o) const {
214         return Z8Number(mulNumbers(digits, o.digits));
215 }
216 
217 Z8Number& Z8Number::operator=(const Z8Number& other) {
218         digits = other.digits;
219         isNegative = other.isNegative;
220         return *this;
221 }
```

```
222
223 std::string Z8Number::toString() const {
224     return digits;
225 }
226
227 void calculate(const Z8Number& a, const Z8Number& b, std::string op) {
228     std::string result;
229     if (!(a.isNegative) && (!(b.isNegative))) {
230         try {
231             if (op == "+") {
232                 result = (a + b).toString();
233             }
234             else if (op == "-") {
235                 try {
236                     result = (a - b).toString();
237                 }
238                 catch (...) {
239                     result = "-" + (b - a).toString();
240                 }
241             }
242             else if (op == "*") {
243                 if (greater(a, b)) result = (a * b).toString();
244                 else result = (b * a).toString();
245             }
246             else if (op == "/") {
247                 result = a.divide(b);
248             }
249             else {
250                 std::cerr << "Unknown operator\n";
251             }
252         }
253         catch (const std::exception& e) {
```

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

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

```
315             else {
316                 std::cerr << "Unknown operator\n";
317             }
318         }
319         catch (const std::exception& e) {
320             std::cerr << "Error: " << e.what() << "\n";
321         }
322     }
323     std::cout << result << std::endl;
324 }
```

Листинг 2: Файл Z8.cpp

Приложение 3. Файл main.cpp.

```
#include <iostream>
#include <string>
#include "z8.h"

int main() {
    std::cout << "Hello!" << std::endl;
    std::string s1, s2, op;
    /*std::cout << "=== DEBUG ===" << std::endl;
    std::cout << "a + b = " << (Z8Number("a") + Z8Number("b")).toString() << "\n";
    std::cout << "f + b = " << (Z8Number("f") + Z8Number("b")).toString() << "\n";
    std::cout << "a * d = " << (Z8Number("a") * Z8Number("d")).toString() << "\n";
    std::cout << "ff * f = " << (Z8Number("ff") + Z8Number("f")).toString() << "\n";
    std::cout << "aaab * c = " << (Z8Number("aaab") + Z8Number("c")).toString() <<
        "\n";
    std::cout << "f - b = " << (Z8Number("f") - Z8Number("b")).toString() << "\n";
    std::cout << "ba - d = " << (Z8Number("ba") - Z8Number("d")).toString() << "\n";
    std::cout << "e + e = " << (Z8Number("e") + Z8Number("e")).toString() << "\n";
    std::cout << "ba / e = " << Z8Number("ba").divide(Z8Number("e")) << "\n";
    std::cout << "-ba + b = "; calculate(Z8Number("-ba"), Z8Number("b"), "+");
    std::cout << "-b + f = "; calculate(Z8Number("-b"), Z8Number("f"), "+");
    std::cout << "-ba - -ff = "; calculate(Z8Number("-ba"), Z8Number("-ff"), "-");
    std::cout << "b - d = "; calculate(Z8Number("b"), Z8Number("d"), "-");
    std::cout << "-a + a = "; calculate(Z8Number("-a"), Z8Number("a"), "+");
    std::cout << "ba * baa = "; calculate(Z8Number("ba"), Z8Number("baa"), "*");
    std::cout << "-a - -a ="; calculate(Z8Number("-a"), Z8Number("-a"), "-");
    std::cout << "-a + -a ="; calculate(Z8Number("-a"), Z8Number("-a"), "+");
    std::cout << "-a * a ="; calculate(Z8Number("-a"), Z8Number("a"), "*");
    std::cout << "-a * baa ="; calculate(Z8Number("-a"), Z8Number("baa"), "*");*/
    while (true) {
```

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

Листинг 3: Файл main.cpp