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).

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**:

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

Также 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 Изоморфизм с кольцом вычетов $\mathbb{Z}/8\mathbb{Z}$

Введём биекцию $\varphi : Z_8 \to \{0, 1, \ldots, 7\}$, сопоставляющую каждому элементу его порядковый номер в последовательности, порождённой многократным применением правила «+1» к элементу a:

$$\varphi(a) = 0, \;\; \varphi(b) = 1, \;\; \varphi(d) = 2, \;\; \varphi(h) = 3,$$
$$\varphi(c) = 4, \;\; \varphi(e) = 5, \;\; \varphi(g) = 6, \;\; \varphi(f) = 7.$$

Отображение φ является изоморфизмом для исходной структуры.

Данное отображение позволяет переводить результаты арифметических операций на числах в восьмеричной системе счисления в исходную структуру (a, b, c...).

2.3 «Большая» конечная арифметика

На основе «малой» арифметики строится позиционная система счисления по основанию 8, в которой:

Цифрами являются элементы множества Z_8,

Числа записываются как строки длиной до 8 символов (например, *bbb*, *ccc*),

Значение числа $X = x_{n-1}x_{n-2}\ldots x_0$ определяется как

$$X = \sum_{k=0}^{n-1} \varphi(x_k) \cdot 8^k. \qquad (1)$$

Арифметические операции над многоразрядными числами реализуются следующим образом:

1. Преобразование операндов в целые числа по формуле (1).

2. Выполнение операции в кольце целых чисел.

3. Преобразование результата обратно в символьную форму с помощью последовательного деления на 8 и применения φ^{-1} к остаткам.

При вычитании из меньшего большего совершается проход по кольцу от наименьшего числа к наибольшему: например, $a - b = fffffffff$.

При нецелом частном ответ записывается в формате "<целая часть>.<остаток>".

Такой подход обеспечивает корректную реализацию всех четырёх арифметических операций $(+, -, \cdot, \div)$ в рамках заданной конечной структуры.

3 Особенности реализации

Программа состоит из двух классов: `Multiset` и `Interface`.

Класс `Multiset` содержит в себе поля для хранения данных о мультимножестве и методы для проведения операций над мультимножествами. Мультимножества хранятся в контейнере `std::string`, т.к. с ним удобно работать.

Класс `Interface` содержит в себе поля для хранения данных о всех мультимножествах в контейнере `std::map<std::string, Multiset>`, где ключи - названия мультимножеств, а значения - сами мультимножества. Методы класса отвечают за консольное меню (пользовательский ввод, вывод мультимножеств и результатов операций над ними).

3.1 Класс `Multiset`

- **Метод** `generateGrayCode`

  Вход: `int n` - разрядность кода Грея.

  Выход: `std::vector<std::string> gray` - контейнер с кодами Грея.

  Назначение: генерирует коды Грея разрядности n.

  Код:

```
1                   if (n <= 0) return {};
2                   std::vector<std::string> gray;
3                   int total = 1 << n;
4
5                   for (int i = 0; i < total; ++i) {
6                       int grayCode = i ^ (i >> 1);
7                       std::string code = "";
8                       for (int j = n - 1; j >= 0; --j) {
9                           code += (grayCode & (1 << j)) ? '1' : '0';
```

```
            }
            gray.push_back(code);
        }
        return gray;
```

- **Метод** `createUniverse`

  Вход: `int depth` - разрядность кода Грея, который будет храниться в универсуме.

  Выход: `std::vector<std::string> universe` - универсум по кодам Грея с заданной разрядностью.

  Назначение: создаёт универсум по кодам Грея.

  Код:

```
void Multiset::createUniverse(int depth) {
    universe = generateGrayCode(depth);
}
```

- **Метод** `setMaxMultiplicity`

  Вход: `int limit` - значение максимальной кратности элементов.

  Выход: `int limit` - значение максимальной кратности элементов.

  Назначение: устанавливает максимальную кратность в универсуме (хранится в виде статической переменной).

  Код:

```
void Multiset::setMaxMultiplicity(int limit) {
    maxMultiplicity = limit;
}
```

- **Метод** `getMaxMultiplicity`

  Вход: `int maxMultiplicity` - значение максимальной кратности элементов.

Выход: `int maxMultiplicity` - значение максимальной кратности элементов.

Назначение: возвращает текущую максимальную кратность.

Код:

```
int Multiset::getMaxMultiplicity() {
    return maxMultiplicity;
}
```

- **Метод** `size`

  Вход: `Multiset input` - мультимножество.

  Выход: `int total` - мощность мультимножества.

  Назначение: возвращает мощность мультимножества.

  Код:

```
int Multiset::size() const {
    int total = 0;
    for (const auto& p : elements) total += p.second;
    return total;
}
```

- **Метод** `fillManual`

  Вход: `int size` - мощность создаваемого мультимножества.

  Выход: `Multiset multi` - мультимножество.

  Назначение: воздание и ручное заполнение мультимножества.

  Код:

```
void Multiset::fillManual(int size) {
    std::cin.ignore();
    if (size < 0) return;

    if (size != 0)
```

```
6                   std::cout << "Enter " << size << " Gray codes (limit: " <<
                        maxMultiplicity << ")\n"
7                   << "(You can enter either Gray code only or Gray code [space]
                        count; other numbers won't be accounted)" << std::endl;
8
9                   for (int i = 0; i < size; ) {
10                      std::string code;
11                      while (true) {
12                          std::cout << "Code #" << (i + 1) << ": ";
13
14                          std::string line;
15                          std::getline(std::cin, line);
16                          std::istringstream iss(line);
17
18                          std::string code;
19                          if (!(iss >> code)) {
20                              std::cout << "Empty input. Try again.\n";
21                              continue;
22                          }
23
24                          int count = 1;
25                          iss >> count;
26
27                          if (!isValidCode(code)) {
28                              std::cout << "Invalid code. Valid: ";
29                              for (const auto& c : universe) std::cout << c << " ";
30                              std::cout << "\nTry again.\n";
31                              continue;
32                          }
33
34                          if (!canAdd(code)) {
35                              std::cout << "You've reached your limit(" <<
```

```
                                maxMultiplicity
36                          << ") for the code \"" << code << "\". Please pick
                                different.\n";
37                          continue;
38                      }
39
40                      int currentCount = frequency(code);
41                      if (currentCount + count > maxMultiplicity) {
42                          std::cout << "Adding " << count << " of \"" <<
                                code
43                          << "\" would exceed limit (" << maxMultiplicity
44                          << "). Currently have " << currentCount << ".\n"
45                          << "Try again with fewer or different code.\n";
46                          continue;
47                      }
48
49                      for (int j = 0; j < count; ++j) {
50                          add(code);
51                      }
52
53                      i += count;
54                      break;
55                  }
56              }
57          }
```

- **Метод** fillRandom

  Вход: int size - мощность создаваемого мультимножества.

  Выход: Multiset multi - мультимножество.

  Назначение: создание и заполнение мультимножества случайными элементами.

Код:

```
void Multiset::fillRandom(int size) {
    if (size <= 0 || universe.empty()) return;

    for (int i = 0; i < size; ++i) {
        std::string code;
        do {
            code = universe[rand() % universe.size()];
        } while (!canAdd(code));

        add(code);
    }
}
```

- **Метод** `frequency`

  Вход: `const std::string& code` - код Грея.

  Выход: `int count` - кратность элемента в мультимножестве.

  Назначение: возвращает кратность элемента (кода Грея) в мультимножестве.

  Код:

```
int Multiset::frequency(const std::string& code) const {
    auto it = elements.find(code);
    return (it != elements.end()) ? it->second : 0;
}
```

- **Метод** `canAdd`

  Вход: `const std::string& code` - код Грея.

  Выход: `bool canAdd` - возможность добавления элемента в мультимножество.

  Назначение: проверяет, можно ли добавить элемент (код Грея) в муль-

тимножество.

Код:

```
bool Multiset::canAdd(const std::string& code) const {
    // Check if code is valid and if adding one more would exceed
        maxMultiplicity
    if (!isValidCode(code)) return false;
    int current = frequency(code);
    return current < maxMultiplicity;
}
```

- **Метод** `isValidCode`

  Вход: `const std::string& code` - код Грея.

  Выход: `bool isValidCode` - корректность данного кода Грея (есть ли он в универсуме).

  Назначение: проверяет корректность данного кода Грея (принадлежит ли код универсуму).

  Код:

```
bool Multiset::isValidCode(const std::string& code) const {
    for (const auto& u : universe) {
        if (code == u) return true;
    }
    return false;
}
```

- **Метод** `add`

  Вход: `const std::string& code, int count = 1` - код Грея, который нужно добавить, и его кратность (по умолч. 1).

  Выход: `int added` - кол-во реально добавленных элементов.

  Назначение: добавляет элементы в массив.

Код:

```
int Multiset::add(const std::string& code, int count = 1) {
    if (!isValidCode(code) || count <= 0) return 0;

    int current = frequency(code);
    int available = maxMultiplicity - current;
    int actualToAdd = std::min(count, available);

    if (actualToAdd > 0) {
        elements[code] += actualToAdd;
    }

    return actualToAdd;
}
```

- **Метод** `unionWith`

  Вход: `Multiset other` - мультимножество, с которым нужно провести операцию объединения.

  Выход: `Multiset result` - мультимножество, результат выполнения операции.

  Назначение: выполнение операции объединения над двумя мультимножествами.

  Код:

```
Multiset Multiset::unionWith(const Multiset& other) const {
    Multiset result;

    for (const auto& p : elements) {
        if (p.second > 0) {
            result.elements[p.first] = p.second;
        }
```

```
        }
        for (const auto& p : other.elements) {
            if (p.second > 0) {
                result.elements[p.first] = std::max(result.elements[p.first],
                    p.second);
            }
        }
        return result;
    }
```

- **Метод** intersectionWith

  Вход: Multiset other - мультимножество, с которым нужно провести операцию пересечения.

  Выход: Multiset result - мультимножество, результат выполнения операции.

  Назначение: выполнение операции пересечения над двумя мультимножествами.

  Код:

```
    Multiset Multiset::intersectionWith(const Multiset& other) const {
        Multiset result;

        for (const auto& p : elements) {
            if (other.elements.count(p.first)) {
                result.elements[p.first] = std::min(p.second, other.elements
                    .at(p.first));
            }
        }
        return result;
    }
```

- **Метод** `differenceWith`

  Вход: `Multiset other` - мультимножество, с которым нужно провести операцию разности.

  Выход: `Multiset result` - мультимножество, результат выполнения операции.

  Назначение: выполнение операции разности над двумя мультимножествами.

  Код:

```
1  Multiset Multiset::differenceWith(const Multiset& other) const {
2      return intersectionWith(other.complement());
3  }
```

- **Метод** `symmetricDifferenceWith`

  Вход: `Multiset other` - мультимножество, с которым нужно провести операцию симметрической разности.

  Выход: `Multiset result` - мультимножество, результат выполнения операции.

  Назначение: выполнение операции симметрической разности над двумя мультимножествами.

  Код:

```
1  Multiset Multiset::symmetricDifferenceWith(const Multiset& other)
       const {
2      Multiset result;
3      auto uni = unionWith(other);
4      auto inter = intersectionWith(other);
5      result = uni.differenceWith(inter);
6
7      return result;
8  }
```

- **Метод** `arithmeticSum`

  Вход: `Multiset other` - мультимножество, с которым нужно провести операцию арифметической суммы.

  Выход: `Multiset result` - мультимножество, результат выполнения операции.

  Назначение: выполнение операции арифметической суммы над двумя мультимножествами.

  Код:

```
1   Multiset Multiset::arithmeticSum(const Multiset& other) const {
2       Multiset result;
3
4       for (const auto& p : elements) result.elements[p.first] += p.second
            ;
5       for (const auto& p : other.elements) result.elements[p.first] += p.
            second;
6       for (auto& p : result.elements)
7       if (p.second > maxMultiplicity)
8       p.second = maxMultiplicity;
9
10      return result;
11  }
```

- **Метод** `arithmeticDifferenceWith`

  Вход: `Multiset other` - мультимножество, с которым нужно провести операцию арифметической разности.

  Выход: `Multiset result` - мультимножество, результат выполнения операции.

  Назначение: выполнение операции арифметической разности над двумя мультимножествами.

Код:

```
Multiset Multiset::arithmeticDifferenceWith(const Multiset& other)
    const {
    Multiset result;

    for (const auto& p : elements) {
        int count = p.second;
        if (other.elements.count(p.first)) {
            count -= other.elements.at(p.first);
        }
        if (count > 0) {
            result.elements[p.first] = count;
        }
    }
    return result;
}
```

- **Метод** `arithmeticProduct`

  Вход: `Multiset other` - мультимножество, с которым нужно провести операцию арифметического произведения.

  Выход: `Multiset result` - мультимножество, результат выполнения операции.

  Назначение: выполнение операции арифметического произведения над двумя мультимножествами.

  Код:

```
Multiset Multiset::arithmeticProduct(const Multiset& other) const {
    Multiset result;

    for (const auto& p : elements) {
        if (other.elements.count(p.first)) {
```

```
6                                   result.elements[p.first] = std::min(p.second * other.
                                        elements.at(p.first), maxMultiplicity);
7                               }
8                           }
9
10                          return result;
11                  }
```

- **Метод** `arithmeticDivision`

  Вход: `Multiset other` - мультимножество, с которым нужно провести операцию арифметического деления.

  Выход: `Multiset result` - мультимножество, результат выполнения операции.

  Назначение: выполнение операции арифметического деления над двумя мультимножествами.

  Код:

```
1                   Multiset Multiset::arithmeticDivision(const Multiset& other) const {
2                       Multiset result;
3
4                       for (const auto& p : elements)
5                       if (other.elements.count(p.first) && other.elements.at(p.first) > 0)
6                       if (p.second / other.elements.at(p.first) > 0)
7                       result.elements[p.first] = p.second / other.elements.at(p.first);
8
9                       return result;
10                  }
```

- **Метод** `complement`

  Вход: `Multiset current` - текущее мультимножество.

  Выход: `Multiset result` - новое мультимножество, являющееся допол-

нением текущего.

Назначение: дополнение относительно максимальной кратности.

Код:

```
Multiset Multiset::complement() const {
    Multiset result;

    for (const auto& code : universe) {
        int current = frequency(code);
        int needed = maxMultiplicity - current;
        if (needed > 0) {
            result.elements[code] = needed;
        }
    }

    return result;
}
```

- **Метод** `print`

  Вход: `Multiset multi` - текущее мультимножество.

  Выход: `Multiset multi` - текущее мультимножество.

  Назначение: печатают мультимножество или универсум.

  Код:

```
void Multiset::print() const {
    std::cout << "{ ";
        bool first = true;
        for (const auto& p : elements) {
            if (!first) std::cout << ", ";
            std::cout << "\"" << p.first << "\"^" << p.second;
            first = false;
        }
        std::cout << "}" << std::endl;
```

```
10          }
```

- **Метод** `printU`

  Вход: `std::vector<std::string> universe` - универсум.

  Выход: `std::vector<std::string> universe` - универсум.

  Назначение: печатает универсум.

  Код:

```
1           void Multiset::printU() const {
2               std::cout << "{ ";
3                   bool first = true;
4                   for (const auto& p : universe) {
5                       if (!first) std::cout << ", ";
6                       std::cout << "\"" << p << "\"^" << 10;
7                       first = false;
8                   }
9                   std::cout << "}" << std::endl;
10          }
```

- **Метод** `getUniverse`

  Вход: `std::vector<std::string> universe` - универсум.

  Выход: `Multiset multiUni` универсум, представленный в виде типа `Multiset`.

  Назначение: возвращает универсум в виде типа `Multiset`.

  Код:

```
1           Multiset Multiset::getUniverse() {
2               Multiset uni;
3               for (const auto& elem : universe) {
4                   uni.elements[elem] = maxMultiplicity;
5               }
6               return uni;
```

```
7                 }
```

3.2 Класс Interface

- **Метод** isValidName

  Вход: const std::string& name - имя нового мультимножества.

  Выход: bool correct - валидно ли имя мультимножества.

  Назначение: проверяет, валидно ли имя мультимножества.

  Код:

```
1                 bool Interface::isValidName(const std::string& name) {
2                     bool isValid = data_.find(name) == data_.end() && name != ""
                          ;
3                     if (isValid)
4                     return true;
5                     else
6                     std::cout << "Multiset with this name already exists or name is
                          invalid.\n"
7                     << "Please, try again: ";
8                     return false;
9                 }
```

- **Метод** createMultisets

  Вход: bool rand - режим создания мультимножества (ручное или случайное).

  Выход: Multiset multi - новое мультимножество.

  Назначение: создаёт новое мультимножество ручным или случайным способом; если нет универсума, сначала создает его.

  Код:

```
1                 void Interface::createMultisets(bool rand) {
```

```
2                       if (data_.size() == 0) { // Here we set the universe parameters
3                           std::cout << "Enter bit depth: ";
4                           int depth;
5                           while (!(std::cin >> depth) || depth < 0) {
6                               std::cout << "Invalid input. Enter non-negative integer: ";
7                               IGN
8                           }
9
10                          bitDepth_ = depth;
11
12                          if (bitDepth_ == 0)
13                          goto AddUniverse;
14
15                          Multiset::createUniverse(depth);
16
17                          std::cout << "Enter universum multiplicity: ";
18                          int multi;
19                          IGN
20                          while (!(std::cin >> multi) || multi < 0) {
21                              std::cout << "Invalid input. Enter non-negative integer: ";
22                              IGN
23                          }
24                          Multiset::setMaxMultiplicity(multi);
25                          uniMultiplicity_ = multi;
26
27                          AddUniverse:
28                          if (uniMultiplicity_ == 0)
29                          Multiset::createUniverse(0);
30                          data_["U"] = Multiset::getUniverse();
31                          uniSize_ = Multiset::getUniverse().size();
32                      }
33
```

```
34                      if (rand) {
35                          createRandom();
36                      }
37                      else {
38                          Multiset A;
39
40                          std::cout << "Enter name of the multiset: ";
41                          std::string name;
42                          std::cin >> name;
43                          while (!isValidName(name)) {
44                              IGN
45                              std::cin >> name;
46                          }
47                          if (bitDepth_ * uniMultiplicity_ == 0) {
48                              goto AddMultiset;
49                          }
50                          std::cout << "\nHow to fill multiset?\n1. Manual\n2. Auto\n
                                > ";
51                          int choice;
52                          IGN
53                          while (!(std::cin >> choice) || (choice != 1 && choice != 2)) {
54                              IGN
55                              std::cout << "Enter 1 or 2: ";
56                          }
57
58                          int sizeA;
59                          std::cout << "Enter size of the multiset " << name << ": ";
60                          IGN
61                          while (!(std::cin >> sizeA) || sizeA < 0 || sizeA > uniSize_) {
62                              IGN
63                              std::cout << "Size can't be negative or more than
                                    universe's size: ";
```

```
64                          }
65
66                          if (choice == 1) {
67                              std::cout << "Fill Multiset " << name << ':' << std::
                                    endl;
68                              A.fillManual(sizeA);
69                          }
70                          else {
71                              A.fillRandom(sizeA);
72                          }
73                          AddMultiset:
74                          data_[name] = A;
75                      }
76                  }
```

- **Метод** displayMenu

  Вход: std::string - текст, который нужно вывести в пользовательском меню.

  Выход: std::string - текст, который нужно вывести в пользовательском меню.

  Назначение: выводит меню программы.

  Код:

```
1                   void Interface::displayMenu() {
2                       std::cout << "Choose an option:\n"
3                       << "1. Create new multisets\n"
4                       << "2. Perform an operation on existing multisets\n"
5                       << "3. Print a multiset\n"
6                       << "4. Print all multisets' names\n"
7                       << "5. Print all multisets\n"
8                       << "6. Create a random multiset\n"
9                       << "7. Delete a multiset\n"
```

```
10                  << "8. Reset (delete all multisets and universe)\n"
11                  << "0. Exit\n> ";
12              }
```

- **Метод** performAll

  Вход: Multiset multi1, multi2; std::string nameA, nameB - мультимножества и их имена.

  Выход: Multiset - результат выполнения операций над мультимножествами.

  Назначение: выполняет все операции и выводит результаты.

  Код:

```
1               void Interface::performAll(Multiset& A, Multiset& B, std::string&
                    nameA, std::string& nameB) {
2                   std::cout << "\n======= MULTISETS =======\n";
3                   std::cout << "U: "; A.printU(); std::cout << '\n';
4                   std::cout << nameA << ": "; A.print(); std::cout << '\n';
5                   std::cout << nameB << ": "; B.print(); std::cout << '\n';
6                   std::cout << "\n======= OPERATIONS =======\n";
7                   std::cout << nameA << " union " << nameB << ": "; A.
                        unionWith(B).print(); std::cout << '\n';
8                   std::cout << nameA << " inter " << nameB << ": "; A.
                        intersectionWith(B).print(); std::cout << '\n';
9                   std::cout << nameA << " \\ " << nameB << ": "; A.
                        differenceWith(B).print(); std::cout << '\n';
10                  std::cout << nameB << " \\ " << nameA << ": "; B.
                        differenceWith(A).print(); std::cout << '\n';
11                  std::cout << nameA << " /\\ " << nameB << ": "; A.
                        symmetricDifferenceWith(B).print(); std::cout << '\n';
12                  std::cout << nameA << " + " << nameB << ": "; A.
                        arithmeticSum(B).print(); std::cout << '\n';
13                  std::cout << nameA << " - " << nameB << ": "; A.
```

```
                arithmeticDifferenceWith(B).print(); std::cout << '\n';
14              std::cout << nameB << " - " << nameA << ": "; B.
                arithmeticDifferenceWith(A).print(); std::cout << '\n';
15              std::cout << nameA << " * " << nameB << ": "; A.
                arithmeticProduct(B).print(); std::cout << '\n';
16              std::cout << nameA << " / " << nameB << ": "; A.
                arithmeticDivision(B).print(); std::cout << '\n';
17              std::cout << nameB << " / " << nameA << ": "; B.
                arithmeticDivision(A).print(); std::cout << '\n';
18              std::cout << "Complement of " << nameA << ": "; auto compA
                    = A.complement(); compA.print(); std::cout << '\n';
19              std::cout << "Complement of " << nameB << ": "; auto compB
                = B.complement(); compB.print(); std::cout << '\n';
20          }
```

- **Метод** `perform`

  Вход: `std::string nameA, nameB` - имена мультимножеств.

  Выход: `Multiset` - результат выполнения операций над мультимножествами.

  Назначение: позволяет выполнять выбранную операцию.

  Код:

```
1           void Interface::perform(std::string& nameA, std::string& nameB) {
2               std::cout << "Enter the name of the 1st multiset: ";
3               std::cin >> nameA;
4               while (isValidName(nameA)) {
5                   std::cout << "Multiset with this name doesn't exist. Please,
                        try again: ";
6                   std::cin >> nameA;
7               }
8               std::cout << "Enter the name of the 2nd multiset: ";
9               std::cin >> nameB;
```

```
10                      while (isValidName(nameB)) {
11                          std::cout << "Multiset with this name doesn't exist. Please,
                                try again: ";
12                          std::cin >> nameB;
13                      }
14
15                      Multiset compA;
16                      Multiset compB;
17                      int trash;
18                      int choice;
19
20                      while (true) {
21                          CL;
22                          std::cout << "Choose an operation:\n"
23                          << "1. '" << nameA << "' union '" << nameB << "'\n"
24                          << "2. '" << nameB << "' inter '" << nameB << "'\n"
25                          << "3. '" << nameA << "' \\ '" << nameB << "'\n"
26                          << "4. '" << nameB << "' \\ '" << nameA << "'\n"
27                          << "5. '" << nameA << "' /\\ '" << nameB << "'\n"
28                          << "6. '" << nameA << "' + '" << nameB << "'\n"
29                          << "7. '" << nameA << "' - '" << nameB << "'\n"
30                          << "8. '" << nameA << "' * '" << nameB << "'\n"
31                          << "9. '" << nameA << "' / '" << nameB << "'\n"
32                          << "0. '" << nameB << "' / '" << nameA << "'\n"
33                          << "d. Do all operations\n"
34                          << "a. Complement of '" << nameA << "'\n"
35                          << "b. Complement of '" << nameB << "'\n"
36                          << "q. Go to main menu\n>";
37
38                          choice = _getch();
39
40                          switch (choice) {
```

```
41                          case '1':
42                          std::cout << "A union B: "; data_[nameA].unionWith(
                                data_[nameB]).print(); std::cout << '\n';
43                          W;
44                          break;
45                          case '2':
46                          std::cout << "A inter B: "; data_[nameA].
                                intersectionWith(data_[nameB]).print(); std::cout <<
                                 '\n';
47                          W;
48                          break;
49                          case '3':
50                          std::cout << "A \\ B: "; data_[nameA].
                                arithmeticDifferenceWith(data_[nameB]).print(); std::
                                cout << '\n';
51                          W;
52                          break;
53                          case '4':
54                          std::cout << "B \\ A: "; data_[nameB].
                                arithmeticDifferenceWith(data_[nameA]).print(); std::
                                cout << '\n';
55                          W;
56                          break;
57                          case '5':
58                          std::cout << "A sym \\ B: "; data_[nameA].
                                symmetricDifferenceWith(data_[nameB]).print(); std::
                                cout << '\n';
59                          W;
60                          break;
61                          case '6':
62                          std::cout << "A + B: "; data_[nameA].arithmeticSum(
                                data_[nameB]).print(); std::cout << '\n';
```

```
63                          W;
64                          break;
65                          case '7':
66                          std::cout << "A - B: "; data_[nameA].differenceWith(
                                data_[nameB]).print(); std::cout << '\n';
67                          W;
68                          break;
69                          case '8':
70                          std::cout << "A * B: "; data_[nameA].arithmeticProduct
                                (data_[nameB]).print(); std::cout << '\n';
71                          W;
72                          break;
73                          case '9':
74                          std::cout << "A / B: "; data_[nameA].arithmeticDivision
                                (data_[nameB]).print(); std::cout << '\n';
75                          W;
76                          break;
77                          case '0':
78                          std::cout << "B / A: "; data_[nameB].arithmeticDivision
                                (data_[nameA]).print(); std::cout << '\n';
79                          W;
80                          break;
81                          case 'd':
82                          performAll(data_[nameA], data_[nameB], nameA, nameB
                                );
83                          W;
84                          break;
85                          case 'a':
86                          std::cout << "Complement of A: "; compA = data_[
                                nameA].complement(); compA.print(); std::cout << '\
                                n';
87                          W;
```

```
88                              break;
89                              case 'b':
90                              std::cout << "Complement of B: "; compB = data_[
                                    nameB].complement(); compB.print(); std::cout << '\
                                    n';
91                              W;
92                              break;
93                              case 'q':
94                              goto Break;
95                              default:
96                              std::cout << "Invalid option. Please, try again!\n";
97                              W;
98                          }
99                          continue;
100                         Break:
101                         break;
102                     }
103                 }
```

- **Метод** printMultiset

  Вход: std::string name - имя мультимножества.

  Выход: Multiset multi - мультимножество.

  Назначение: выводит содержимое множества.

  Код:

```
1               void Interface::printMultiset(std::string& name) {
2                   data_[name].print();
3               }
```

- **Метод** printNames

  Вход: std::string - имена мультимножеств.

Выход: `std::string` - имена мультимножеств.

Назначение: выводит список имён всех множеств.

Код:

```
void Interface::printNames() {
    int i = 1;
    for (const auto& pair : data_) {
        std::cout << "#" << i << ": " << pair.first << std::endl;
        i++;
    }
}
```

- **Метод** `printAllMultisets`

  Вход: `Multiset` - все созданные мультимножества.

  Выход: `Multiset` - все созданные мультимножества.

  Назначение: печатает все мультимножества.

  Код:

```
void Interface::printAllMultisets() {
    int i = 1;
    for (const auto& pair : data_) {
        std::cout << '#' << i << " \"" << pair.first << "\": ";
        pair.second.print();
        i++;
    }
}
```

- **Метод** `createRandom`

  Вход: `Multiset universe` - универсум, на основе которого будет создаваться мультимножество.

  Выход: `Multiset multi` - мультимножество.

Назначение: создаёт случайное мультимножество.

Код:

```
void Interface::createRandom() {
    std::string name;
    int i = 10000;
    do { name = std::to_string(rand() % i++); } while (!isValidName
        (name));
    Multiset A;
    i = 100;
    int size = rand() % i;
    while (size > uniMultiplicity_ * bitDepth_)
    size--;
    A.fillRandom(size);
    data_[name] = A;
    std::cout << "Random multiset created.\n"
    << "Name: " << name
    << "\nSize: " << A.size() << std::endl;
}
```

- **Метод** `deleteMultiset()`

  Вход: `std::string name` - имя удаляемого мультимножества.

  Выход: `std::map<std::string, Multiset> data` - контейнер мультимножеств (обновленный: удалено выбранное мультимножество).

  Назначение: удаляет мультимножество.

  Листинг:

```
void Interface::deleteMultiset() {
    std::string name;
    std::cout << "Enter the name of the multiset to delete: ";
    std::cin >> name;
    while (isValidName(name) || name == "U") { // Checks if it
        DOESN'T exist yet.
```

```
6                       if (isValidName(name))
7                       std::cout << "Multiset with this name doesn't exist. Please,
                            try again: ";
8                       else
9                       std::cout << "Can't delete universe. Please, try again: ";
10                      std::cin >> name;
11                  }
12                  data_.erase(name);
13                  std::cout << "Deleted." << std::endl;
14              }
```

- **Метод** `reset`

  Вход: `std::map<std::string, Multiset> data` - контейнер мультимножеств.

  Выход: `std::map<std::string, Multiset> data` - пустой контейнер мультимножеств.

  Назначение: удаляет все мультимножества.

  Листинг:

```
1               void Interface::reset() {
2                   data_.clear();
3                   bitDepth_ = 0;
4                   uniMultiplicity_ = 0;
5               }
```

- **Метод** `run`

  Вход: `std::iostream` - поля ввода и вывода программы.

  Выход: `int` - статус завершения работы программы.

  Назначение: запускает основной цикл программы.

  Листинг:

```
int Interface::run() {
    int trash;
    std::string nameA;
    std::string nameB;

    std::cout << "Hello!\n"
    << "Here you can create multisets and perform operations upon
        them.\n"
    << "Press any key to continue.";
    trash = _getch();

    while (true) {
        CL;
        displayMenu();
        Again:
        char option = _getch();
        switch (option) {
            case '1':
            CL;
            createMultisets();
            std::cout << "Press any key to continue.";
            trash = _getch();
            break;

            case '2':
            CL;
            if (data_.size() < 2) {
                std::cout << "You need at least two multisets to
                    perform operations.\n"
                << "Press any key to continue.";
                trash = _getch();
                break;
```

```
31                          }
32                          perform(nameA, nameB);
33                          break;
34
35                          case '3':
36                          CL;
37                          if (data_.size() == 0) {
38                              std::cout << "Nothing to print. No multisets available
                                    ." << std::endl;
39                              W;
40                              break;
41                          }
42                          std::cout << "Enter the name of a multiset to print: ";
43                          std::cin >> nameA;
44                          while (isValidName(nameA)) { // Checks if it DOESN'T
                                exist yet.
45                              std::cout << "Wrong name. Please try again: ";
46                              std::cin >> nameA;
47                          }
48                          printMultiset(nameA);
49                          W;
50                          break;
51
52                          case '4':
53                          CL;
54                          if (data_.size() == 0) {
55                              std::cout << "Nothing to print. No multisets available
                                    ." << std::endl;
56                              W;
57                              break;
58                          }
59                          printNames();
```

```
60                          W;
61                          break;
62
63                          case '5':
64                          CL;
65                          if (data_.size() == 0) {
66                              std::cout << "Nothing to print. No multisets available
                                    ." << std::endl;
67                              W;
68                              break;
69                          }
70                          printAllMultisets();
71                          W;
72                          break;
73
74                          case '6':
75                          CL;
76                          if (data_.size() == 0)
77                          createMultisets(true);
78                          else
79                          createRandom();
80                          W;
81                          break;
82
83                          case '7':
84                          CL;
85                          if (data_.size() == 0) {
86                              std::cout << "Nothing to delete. No multisets
                                    available." << std::endl;
87                              W;
88                              break;
89                          }
```

```
90                              deleteMultiset();
91                              W;
92                              break;
93
94                              case '8':
95                              CL;
96                              reset();
97                              std::cout << "Resetted." << std::endl;
98                              W;
99                              break;
100
101                             case '0':
102                             return 0;
103
104                             default:
105                             std::cout << "Invalid option. Please, try again!\n";
106                             goto Again;
107                         }
108                     }
109                 }
```

%enditemize

4 Результаты работы программы

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

Меню программы

После запуска программы отображается главное меню, через которое пользователь может:

- создать новое мультимножество (ручное или случайное заполнение);
- вывести список всех созданных мультимножеств;
- выбрать два мультимножества для выполнения операций;
- выполнить все операции сразу или выбрать отдельную операцию;
- выйти из программы.

Создание мультимножеств

Программа поддерживает два режима создания мультимножеств:

- **ручной режим**: пользователь вводит коды Грея и их кратности, программа проверяет корректность кода и не позволяет превысить максимальную кратность;
- **автоматический режим**: программа случайным образом заполняет мультимножество элементами универсума, не превышая заданные ограничения; пользователю необходимо ввести имя и мощность.

  Также для быстрого создания мультимножеств можно использовать создание случайного мультимножества: в таком случае у него будут случайные имя и содержимое, что особенно удобно для быстрой генерации.

Работа с несколькими мультимножествами

Программа может одновременно хранить много мультимножеств, каждому из которых присваивается уникальное имя. В любой момент пользователь может вывести названия и содержимое всех мультимножеств. Это особенно удобно, если пользователь забыл их названия: с помощью этого списка он сможет найти нужные ему мультимножества и провести над ними необходимые операции.

Обработка некорректного ввода

Программа устойчиво ведёт себя при некорректных данных:

- При вводе неверных команд в меню отображается предупреждение и предлагается повторить ввод;
- при попытке ввести код, отсутствующий в универсуме, выводится сообщение об ошибке, а также список возможных кодов (сам универсум);
- при попытке превысить максимальную кратность программа блокирует добавление;
- при попытке обращения к несуществующему мультимножеству отображается предупреждение и предлагается повторить ввод;
- при попытке создать мультимножество с уже существующим именем программа предлагает повторить ввод и т.д.

Работа с пустыми множествами

Программа корректно обрабатывает пустые мультимножества: все операции (объединение, пересечение, разность и др.) с пустыми мультимно-

жествами дают ожидаемый математический результат, что демонстрирует устойчивость реализации.

Пример работы

При запуске программы высвечивается приветственное окно. Чтобы продолжить в главное меню и начать пользоваться основным функционалом программы, надо нажать на любую кнопку (рис. 1).

Рис. 1: Приветственное сообщение при запуске программы.

После этого пользователь попадает в главное меню, где ему предлагается ряд опций на выбор. Чтобы выбрать нужную опцию, пользователю нужно ввести соответствующую цифру (рис. 2).

Рис. 2: Меню программы.

Если пользователь введет некорректный вариант, программа об этом сообщит и предложит выбрать снова (рис. 3).

Рис. 3: Обработка некорретного пользовательского ввода.

Если пользователь решит выполнить какие-то операции над мультимножествами, пока они не заданы, программа сообщит об этом и попросит сначала создать мультимножества (рис. 4).

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

Процесс создания мультимножеств включает в себя несколько этапов (рис. 5).

1. Ввод разрядности, если универсум еще не создан (например, 3).
2. Ввод кратности элементов универсума, если он еще не создан (например, 5).
3. Название мультимножества (например, A).
4. Выбор заполнения мультимножества: ручной или автоматический.
5. Ввод мощности мультимножества (например, 10).

Рис. 5: Создание мультимножества с автоматическим заполнением.

При ручном наборе элементов мультимножества пользователь может вводить как указать кратность текущего элемента, так и опустить ее (в таком случае будет использовано значение кратности по умолчанию - 1). Если такой элемент уже существует, кратности будут сложены. Если в процессе создания мультимножества будут введены неверные данные (например, слишком большая кратность), программа сообщит об этом и попросит ввести корретные данные (рис. 6).

Рис. 6: Создание мультимножества с ручным вводом элементов.

После того как пользователь создаст хотя бы одно мультимножество, он сможет использовать весь функционал программы, в том числе операции над ними, поскольку вместе с первым мультимножеством создается и универсум, который можно увидеть, например, при выборе опции 5 "Print all multisets"(рис. 7).

Рис. 7: Вывод всех мультимножеств.

Для проведения операций над мультимножествами пользователю надо выбрать пункт 2 в меню (рис. 8).

Рис. 8: Опция для проведения операций над мультимножествами.

Далее ему будет предложено ввести названия мультимножеств, с которыми будут проводиться операции. При неверном названии будет предупреждение от программы и просьба ввести корректное название (рис. 9).

Рис. 9: Ввод названий мультимножеств для проведения операций.

После ввода названий мультимножеств пользователю предоставляется список возможных операций над ними. Чтобы выбрать операцию, пользователю нужно нажать на соответствующую клавишу. При неверном выборе программа попросит пользователя выбрать корректный вариант (рис. 10).

Рис. 10: Неверный выбор операции над мультимножествами.

Для удобства пользователю предоставляется вариант "Do all operations"("Сделать все операции"). При выборе этого варианта пользователь получит результаты всех предложенных операций над этими мультимножествами (рис. 11).

Рис. 11: Выполнение всех операций над мультимножествами.

Результаты подтвердили, что все операции над мультимножествами реализованы корректно.

Заключение

В ходе лабораторной работы была разработана программа, позволяющая:

- генерировать бинарный код Грея заданной разрядности;
- формировать мультимножества двумя способами (ручной и автоматический);
- выполнять над мультимножествами различные операции:
  - объединение,
  - пересечение,
  - разность,
  - симметрическая разность,
  - дополнение,
  - арифметическая сумма,
  - арифметическая разность,
  - арифметическое произведение,
  - арифметическое деление;
- обеспечивать защиту от некорректного пользовательского ввода.

Программа демонстрирует корректную реализацию всех заданных функций. Программа создана по принципам ООП: выделены отдельные классы `Multiset` и `Interface`, что удобно разделяет интерфейс и функционал каждого из класса 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.
- Возможность работы над несколькими мультимножествами без стирания предыдущих из памяти.
- Возможность удаления мультимножеств.

Минусы программы.

- Все данные хранятся в оперативной памяти, без сохранения на диск.
- Ограничения по размеру входных данных (из-за возможного целочисленного переполнения при больших объемах).
- Использование контейнеров STL, что ухудшает производительность и повышает расходы памяти.

Масштабирование. Используемая архитектура позволяет легко добавлять новые операции над мультимножествами (например, вычисление мощности или сортировку по частоте). Также возможна интеграция с графическим интерфейсом или веб-интерфейсом. В будущем можно добавить возможность работы с тернарными операциями.

В перспективе программа может быть:

- использована как часть библиотеки по дискретной математике;
- интегрирована в учебные курсы для демонстрации операций над мультимножествами;

- адаптирована под параллельные вычисления для одновременных операций над большими мультимножествами или для ускорения самих операций.

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] Коды Грэя и задачи перебора (Электронный ресурс).
URL: https://habr.com/ru/articles/200806/
(дата обращения: 13.09.2025).

[3] Секция "Телематика"(Электронный ресурс).
URL: https://tema.spbstu.ru/dismath/
(дата обращения: 13.09.2025).

[4] Класс map | Microsoft Learn (Электронный ресурс).
URL: https://learn.microsoft.com/ru-ru/cpp/standard-library/map-class?view=msvc-170
(дата обращения: 18.09.2025).

Приложение 1. Файл Multiset.h.

Приложение 2. Файл Multiset.cpp.

Приложение 3. Файл Interface.h.

Приложение 4. Файл Interface.cpp.

Приложение 5. Файл main.cpp.