

МИНОБРНАУКИ РОССИИ

«Санкт-Петербургский политехнический университет  
Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Отчет о выполнении лабораторной работы №1

Дискретная математика

«Реализация генерации бинарного кода Грея и операций над  
мультимножествами на его основе»

Студент,

группа 5130201/30002

\_\_\_\_\_ Невечерин А. А.

Преподаватель

\_\_\_\_\_ Востров А. В.

« \_\_\_\_\_ » \_\_\_\_\_ 2025г.

Санкт-Петербург  
2025

# Содержание

<b>Введение</b>	<b>4</b>
<b>1 Математическое описание</b>	<b>5</b>
1.1 Множества . . . . .	5
1.2 Мультимножества . . . . .	5
1.3 Операции над множествами . . . . .	5
1.3.1 Объединение . . . . .	5
1.3.2 Пересечение . . . . .	5
1.3.3 Разность . . . . .	5
1.3.4 Дополнение . . . . .	6
1.3.5 Симметрическая разность . . . . .	6
1.3.6 Арифметическая сумма . . . . .	6
1.3.7 Арифметическая разность . . . . .	6
1.3.8 Арифметическое произведение . . . . .	6
1.3.9 Арифметическое деление . . . . .	7
1.4 Код Грея . . . . .	7
<b>2 Особенности реализации программы</b>	<b>8</b>
2.1 Класс BaseSet . . . . .	8
2.1.1 Конструктор . . . . .	8
2.1.2 SetUnion . . . . .	8
2.1.3 SetIntersection . . . . .	9
2.1.4 operator! . . . . .	10
2.1.5 SetDifference . . . . .	11
2.1.6 SetXor . . . . .	11
2.1.7 operator+ . . . . .	11
2.1.8 operator- . . . . .	12
2.1.9 operator* . . . . .	13
2.1.10 operator/ . . . . .	14
2.1.11 operator« . . . . .	15
2.1.12 GetGrayFunctionElement . . . . .	16
2.1.13 GrayPodFunction . . . . .	16
2.2 Классы наследники BaseSet . . . . .	17
2.2.1 Конструктор класса Universum . . . . .	17
2.2.2 Конструктор класса EmptySet . . . . .	17
2.2.3 Конструктор класса MultiSet №1 . . . . .	18
2.2.4 Конструктор класса MultiSet №2 . . . . .	20
2.3 Класс Screen . . . . .	20
2.3.1 Конструктор . . . . .	20
2.3.2 ProgramStart . . . . .	20
2.3.3 GenerateMultisetMenu . . . . .	21
2.3.4 GenerateRandomMultiSet . . . . .	22
2.3.5 GenerateMultiSet . . . . .	23
2.3.6 PrintMainMenu . . . . .	23
2.3.7 PrintActionMenu . . . . .	24
2.3.8 Меню действий . . . . .	25
<b>3 Результаты работы программы</b>	<b>26</b>

Заключение	34
Список литературы	36

# Введение

Данная лабораторная работа заключается в разработке программы, которая генерирует бинарный код Грея для заполнения универсума мультимножества, заданной пользователем разрядности. На основе универсума формируются два мультимножества, двумя возможными способами заполнения - вручную и автоматически, по выбору пользователя. Пользователь также задает мощности и количество ненулевых элементов множеств.

В результате выполнения программы на экран выводятся результаты действий над множествами: объединение, пересечение, разность (оба варианта), симметрическая разность, дополнение (оба варианта), а также арифметические сумма, разность, произведение и деление.

Кроме того программа включает защиту от некорректного ввода данных пользователем и имеет удобное пользовательское меню.

# 1 Математическое описание

## 1.1 Множества

Понятие множества принадлежит к числу фундаментальных понятий математики. Можно сказать, что множество — это любая определённая совокупность объектов. Объекты, из которых составлено множество, называются его элементами. Элементы множества различны и отличимы друг от друга. Как множествам, так и элементам можно давать имена или присваивать символьные обозначения.

Если объект  $x$  является элементом множества  $M$ , то говорят, что  $x$  принадлежит  $M$ . Обозначение:  $x \in M$ . В противном случае говорят, что  $x$  не принадлежит  $M$ . Обозначение:  $x \notin M$ .

Множество, не содержащее элементов, называется пустым. Обозначение:  $\emptyset$ .

Обычно в конкретных рассуждениях элементы всех рассматриваемых множеств берутся из некоторого одного, достаточно широкого множества  $U$  (своего для каждого случая), которое называется универсальным множеством (или универсумом).

## 1.2 Мультимножества

В множестве все элементы различны, а значит, входят в множество ровно один раз. Мультимножество же это совокупность элементов, в которую элементы входят по несколько раз.

Пусть  $X = \{x_i, \dots, x_n\}$  — некоторое (конечное) множество и пусть  $a_1, \dots, a_n$  — неотрицательные целые числа. Тогда *мультимножеством*  $\hat{X}$  (над множеством  $X$ ) называется совокупность элементов множества  $X$ , в которую элемент  $x_i$  входит  $a_i$  раз,  $a_i \geq 0$ . Мультимножество обозначается одним из следующих способов:

$$X = [x_1^{a_1}, \dots, x_n^{a_n}] \text{ или } X = \langle a_1(x_1), \dots, a_n(x_n) \rangle.$$

## 1.3 Операции над множествами

### 1.3.1 Объединение

Результатом объединения множеств является такое множество, которое содержит в себе все элементы исходных множеств. В результирующем мультимножестве кратность каждого элемента равняется максимальной кратности соответствующего элемента в объединяемых Множествах  $A$  и  $B$ .

$$\text{Множества: } C = A \cup B = \{x | x \in A \vee x \in B\}$$

$$\text{Мультимножества: } C = A \cup B = \{\max(a_i(x_i), a_j(x_j))\}, a_i(x_i) \in A, a_j(x_j) \in B$$

### 1.3.2 Пересечение

Результатом пересечения множеств является такое множество, которому принадлежат те и только те элементы, которые одновременно принадлежат всем данным множествам. В результирующем мультимножестве кратность каждого элемента равняется минимальной кратности соответствующего элемента в пересекаемых Множествах  $A$  и  $B$ .

$$\text{Множества: } C = A \cap B = \{x | x \in A \wedge x \in B\}$$

$$\text{Мультимножества: } C = A \cap B = \{\min(a_i(x_i), a_j(x_j))\}, a_i(x_i) \in A, a_j(x_j) \in B$$

### 1.3.3 Разность

Результатом разности множеств является такое множество, в которое входят все элементы первого множества, не входящие во второе множество. В результирующем мульти-

множестве кратность каждого элемента равняется минимальной кратности соответствующего элемента в пересекающихся множествах  $A$  и  $\overline{B}$ .  $A \setminus B = (A \cap \overline{B})$

Множества:  $C = A \setminus B = \{x | x \in A \wedge x \notin B\}$

Мультимножества:  $C = A \setminus B = \{\min(a_i(x_i), a_j(x_j))\}, a_i(x_i) \in A, a_j(x_j) \in \overline{B}$

### 1.3.4 Дополнение

Результатом дополнения множества является новое множество, элементы которого не принадлежат данному множеству. Данная операция подразумевает, что существует универсум. В результирующем мультимножестве кратность каждого элемента равняется разности кратностей соответствующих элементов в универсуме.  $\overline{A} = U \setminus A$

Множества:  $C = \overline{A} = \{x | x \notin A\}$

Мультимножества:  $C = \overline{A} = \{\max(a_i(x_i) - a_j(x_j), 0)\}, a_i(x_i) \in U, a_j(x_j) \in A$

### 1.3.5 Симметрическая разность

Результатом симметрической разности множеств является новое множество, включающее все элементы исходных множеств, не принадлежащие одновременно обоим исходным множествам. В результирующем мультимножестве кратность каждого элемента равняется модулю разности кратностей соответствующих элементов в вычитаемых мультимножествах.  $A \triangle B = (A \cup B) \setminus (A \cap B)$

Множества:  $C = A \triangle B = \{x | x \in A \wedge x \notin B\} \vee \{x | x \notin A \wedge x \in B\}$

Мультимножества:  $C = A \triangle B = \{a(x) : a(x) = |a_i(x_i) - a_j(x_j)|\}, a_i(x_i) \in A, a_j(x_j) \in B$

### 1.3.6 Арифметическая сумма

Результатом арифметической суммы множеств является новое множество, которое содержит в себе все элементы исходных множеств. В результирующем мультимножестве кратность каждого элемента равняется сумме кратностей соответствующих элементов в складываемых мультимножествах, но эта кратность не может быть больше соответствующей кратности универсума.

Множества:  $C = A + B = \{x | x \in A \vee x \in B\}$

Мультимножества:  $C = A + B = \{\min(a_i(x_i) + a_j(x_j), u_k(x_k))\}, a_i(x_i) \in A, a_j(x_j) \in B, u_k(x_k) \in U$

### 1.3.7 Арифметическая разность

Результатом арифметической разности множеств является новое множество, в которое входят все элементы первого множества, не входящие во второе множество. В результирующем мультимножестве кратность каждого элемента равняется разности кратностей соответствующих элементов, и равна нулю, если кратность элемента из первого мультимножества меньше кратности соответствующего элемента из второго.

Множества:  $C = A - B = \{x | x \in A \wedge x \notin B\}$

Мультимножества:  $C = A - B = \{\max(a_i(x_i) - a_j(x_j), 0)\}, a_i(x_i) \in A, a_j(x_j) \in B$

### 1.3.8 Арифметическое произведение

Результатом произведения множеств является новое множество, которое содержит те и только те элементы, которые одновременно принадлежат всем данным множествам. В результирующем мультимножестве кратность каждого элемента равняется произведению кратностей соответствующих элементов, но эта кратность не может быть больше соответствующей кратности элемента в универсуме.

Множества:  $C = A * B = \{x | x \in A \wedge x \in B\}$

Мультимножества:  $C = A * B = \{\min(a_i(x_i) * a_j(x_j), u_k(x_k))\}, a_i(x_i) \in A, a_j(x_j) \in B, u_k(x_k) \in U$

### 1.3.9 Арифметическое деление

Результатом деления множеств является новое множество, которое содержит в себе те и только те элементы, которые одновременно принадлежат всем данным множествам. В результирующем мультимножестве кратность каждого элемента равняется частному кратностей соответствующих элементов, кратность это положительное целое число. В случае, когда происходит деление на 0, то есть кратность элемента во втором множестве равна нулю, то в результате кратность соответствующего элемента будет равна 0.

Множества:  $C = A : B = \{x | x \in A \wedge x \in B\}$

Мультимножества:  $C = A : B = \{a(x) : a_j(x_j) \neq 0 \ \& \ a(x) = \max(a_i(x_i) : a_j(x_j), 0) \ \vee \ a_j(x_j) = 0 \ \& \ a(x) = 0\}, a_i(x_i) \in A, a_j(x_j) \in B$

## 1.4 Код Грея

Код Грея – двоичный код, использующий символы 0 и 1, в котором две соседние кодовые комбинации различаются между собой значением символа только в одном разряде.

Алгоритм работает следующим образом: сначала создается массив В длиной n, который будет использоваться для представления множеств. Он заполняется нулями, после чего выполняется цикл от 1 до  $(2^n - 1)$ . В каждой итерации вызывается функция Q, определяющая индекс, который нужно добавить или удалить в массиве В, основываясь на двоичном представлении текущего числа i. Затем элемент В[p] инвертируется (0 становится 1, и наоборот), и обновленный массив В возвращается.

Алгоритм генерации кода Грея представлен ниже:

```
1 Input: n >= 0
2 Output: B
3   B: array [1 ... n] of 0 ... 1
4   for i from 1 to n
5     do B[i]: = 0
6   end for
7   yield B
8   for i from 1 to 2^n - 1 do
9     p: = Q(i)
10    B[p]: = 1 - B[p]
11    yield B
12 end for
13
14 Function Q
15 Input: i
16 Output: q
17   q: = 1; j: = i
18   while (j mod 2 = 0) do
19     j: = j / 2; q: = q + 1
20   end while
21   return q
```

## 2 Особенности реализации программы

В этом разделе будет подробно рассмотрена структура разработанного приложения. Отдельное внимание будет уделено его классовой архитектуре, т.е. будет показано какие классы были созданы, для чего и т.д. Также будет рассмотрена иерархия классов, их атрибуты и методы, что позволит лучше понять, как они взаимодействуют друг с другом для выполнения поставленных задач.

### 2.1 Класс BaseSet

Класс BaseSet представляет из себя базовый класс, от которого наследуются классы MultiSet, Universum и EmptySet. Класс реализует весь основной функционал мультимножеств, а так же в нем описаны основные характеристики мультимножеств, такие как порядность элементов, мощность и операции над мультимножествами. Основным полем класса является вектор, хранящий в себе значения кратностей элементов множества. Заполнение множества кратностями будет происходить в конструкторах производных классов.

#### 2.1.1 Конструктор

Вход: разрядность элементов, создаваемого множества и его имя.

Выход: создание экземпляра множества BaseSet.

Конструктор создает экземпляр множества с заданным именем и размерностью элементов. Мощность множества устанавливается в 0.

```
1 BaseSet::BaseSet(short elements_size, string set_name) {
2     this->name = set_name;
3     this->elements_size = elements_size;
4     this->power = 0;
5 }
```

#### 2.1.2 SetUnion

Вход: 2 мультимножества и название нового мультимножества, которое будет получено после выполнения действия объединения.

Выход: новое мультимножество - результат выполнения объединения для двух переданных мультимножеств.

Функция проверяет, является ли какое-либо из переданных множеств пустым (объектом класса EmptySet). Если оба множества пустые, то она возвращает пустое множество (указатель на новый объект EmptySet). Если только одно из множеств пустое, то она возвращает второе непустое множество (указатель на объект MultiSet). Если же оба множества непустые, то результатом является новое мультимножество, кратности элементов которого получаются согласно правилу, по которому выполняется объединение мультимножеств. Функция возвращает указатель на новый объект MultiSet, в конструктор которого передаются полученные в результате выполнения действия кратности.

```
1 unique_ptr<BaseSet> SetUnion(BaseSet& set1, BaseSet& set2, string
2     set_name) {
3     EmptySet* emptyset1 = dynamic_cast<EmptySet*>(&set1);
4     EmptySet* emptyset2 = dynamic_cast<EmptySet*>(&set2);
5     if (emptyset1 && emptyset2) {
6         return make_unique<EmptySet>(emptyset1->GetUniversum(), set_name);
7     }
8     else if (emptyset1) {
```



```

8     MultiSet* multiset2 = dynamic_cast<MultiSet*>(&set2);
9     if (multiset2) return make_unique<MultiSet>(*multiset2, set_name);
10 }
11 else if (emptyset2) {
12     MultiSet* multiset1 = dynamic_cast<MultiSet*>(&set1);
13     if (multiset1) return make_unique<MultiSet>(*multiset1, set_name);
14 }
15
16 vector<int> multiplicities;
17 bool zero_flag = true;
18 for (int i = 0; i < pow(2, set1.elements_size); i++) {
19     multiplicities.push_back(max(set1[i], set2[i]));
20     if (zero_flag && max(set1[i], set2[i]) != 0) {
21         zero_flag = false;
22     }
23 }
24
25 MultiSet* multiset1 = dynamic_cast<MultiSet*>(&set1);
26 MultiSet* multiset2 = dynamic_cast<MultiSet*>(&set2);
27 if (multiset1) {
28     if (zero_flag) return
29         make_unique<EmptySet>(multiset1->GetUniversum(), set_name);
30     return make_unique<MultiSet>(multiset1->GetUniversum(), set_name,
31         multiplicities);
32 }
33 if (multiset2) {
34     if (zero_flag) return
35         make_unique<EmptySet>(multiset2->GetUniversum(), set_name);
36     return make_unique<MultiSet>(multiset2->GetUniversum(), set_name,
37         multiplicities);
38 }
39 }

```

### 2.1.3 SetIntersection

Вход: 2 мультимножества и название нового мультимножества, которое будет получено после выполнения действия пересечения.

Выход: новое мультимножество - результат выполнения пересечения для двух переданных мультимножеств.

Функция проверяет, является ли какое-либо из переданных множеств пустым (объектом класса EmptySet). Если хотя бы одно из множеств пустое, то она возвращает пустое множество (указатель на новый объект EmptySet). Если же оба множества непустые, то результатом является новое мультимножество, кратности элементов которого получаются согласно правилу, по которому выполняется пересечение мультимножеств. Результат выполнения действия может являться пустым множеством и тогда функция вернет указатель на новый объект EmptySet. Иначе функция возвращает указатель на новый объект MultiSet, в конструктор которого передаются полученные в результате выполнения действия кратности.

```

1 unique_ptr<BaseSet> SetIntersection(BaseSet& set1, BaseSet& set2, string
2     set_name) {
3     EmptySet* emptyset1 = dynamic_cast<EmptySet*>(&set1);
4     EmptySet* emptyset2 = dynamic_cast<EmptySet*>(&set2);
5     if (emptyset1 || emptyset2) {

```

```

5         if (emptyset1) return
           make_unique<EmptySet>(emptyset1->GetUniversum(), set_name);
6     return make_unique<EmptySet>(emptyset2->GetUniversum(), set_name);
7     }
8
9     vector<int> multiplicities;
10    bool zero_flag = true;
11    for (int i = 0; i < pow(2, set1.elements_size); i++) {
12        multiplicities.push_back(min(set1[i], set2[i]));
13        if (zero_flag && min(set1[i], set2[i]) != 0) {
14            zero_flag = false;
15        }
16    }
17
18    MultiSet* multiset1 = dynamic_cast<MultiSet*>(&set1);
19    MultiSet* multiset2 = dynamic_cast<MultiSet*>(&set2);
20    if (multiset1) {
21        if (zero_flag) return
           make_unique<EmptySet>(multiset1->GetUniversum(), set_name);
22        return make_unique<MultiSet>(multiset1->GetUniversum(), set_name,
           multiplicities);
23    }
24    if (multiset2) {
25        if (zero_flag) return
           make_unique<EmptySet>(multiset2->GetUniversum(), set_name);
26        return make_unique<MultiSet>(multiset2->GetUniversum(), set_name,
           multiplicities);
27    }
28    }

```

#### 2.1.4 operator!

Вход: мультимножество.

Выход: новое мультимножество - результат выполнения действия дополнения для переданного мультимножества.

Метод проверяет, является ли переданное множество пустым (объектом класса EmptySet). Если множество пустое, то он возвращает универсум (т.к. реализация кода подразумевает существование только одного экземпляра класса Universum, то метод возвращает указатель на новый объект MultiSet, все кратности которого равны кратностям универсума). Если же множество непустое, то результатом является новое мультимножество, кратности элементов которого получаются согласно правилу, по которому выполняется дополнение мультимножества. Метод возвращает указатель на новый объект MultiSet, в конструктор которого передаются полученные в результате выполнения действия кратности.

```

1    unique_ptr<BaseSet> operator! (BaseSet& set) {
2        string set_name = "!" + set.name;
3
4        EmptySet* emptyset1 = dynamic_cast<EmptySet*>(&set);
5        if (emptyset1) return
           make_unique<MultiSet>(emptyset1->GetUniversum(), set_name);
6
7        Universum* universum1 = dynamic_cast<Universum*>(&set);
8        if (universum1) return make_unique<EmptySet>(*universum1, set_name);
9    }

```

```

10     MultiSet* multiset1 = dynamic_cast<MultiSet*>(&set);
11     vector<int> multiplicities;
12     vector<int> universum_vector =
        multiset1->GetUniversum().GetMainVectorCopy();
13     for (int i = 0; i < pow(2, set.elements_size); i++) {
14         multiplicities.push_back(universum_vector[i] - set.main_vector[i]);
15     }
16
17     return make_unique<MultiSet>(multiset1->GetUniversum(), set_name,
        multiplicities);
18 }

```

### 2.1.5 SetDifference

Вход: 2 мультимножества (A и B) и название нового мультимножества, которое будет получено после выполнения действия разности.

Выход: новое мультимножество - результат выполнения разности для двух переданных мультимножеств.

Т.к.  $A \setminus B = (A \cap \overline{B})$ , то для выполнения данного действия достаточно вернуть результат выполнения функции SetIntersection для множества A и дополнения множества B.

```

1     unique_ptr<BaseSet> SetDifference(BaseSet& set1, BaseSet& set2, string
        set_name) {
2         return SetIntersection(set1, *!set2, set_name);
3     }

```

### 2.1.6 SetXor

Вход: 2 мультимножества (A и B) и название нового мультимножества, которое будет получено после выполнения действия симметрической разности.

Выход: новое мультимножество - результат выполнения симметрической разности для двух переданных мультимножеств.

Т.к.  $A \triangle B = (A \cup B) \setminus (A \cap B)$ , то для выполнения данного действия достаточно вернуть результат выполнения функции SetDifference для функции SetUnion(A, B) и функции SetIntersection(A, B).

```

1     unique_ptr<BaseSet> SetXor(BaseSet& set1, BaseSet& set2, string
        set_name) {
2         return SetDifference(*SetUnion(set1, set2), *SetIntersection(set1,
        set2), set_name);
3     }

```

### 2.1.7 operator+

Вход: 2 мультимножества и название нового мультимножества, которое будет получено после выполнения действия арифметического сложения.

Выход: новое мультимножество - результат выполнения арифметического сложения для двух переданных мультимножеств.

Метод проверяет, является ли какое-либо из переданных множеств пустым (объектом класса EmptySet). Если оба множества пустые, то он возвращает пустое множество (указатель на новый объект EmptySet). Если только одно из множеств пустое, то метод возвращает второе непустое множество (указатель на объект MultiSet). Если же оба множества непустые, то результатом является новое мультимножество, кратности элементов которого

получаются согласно правилу, по которому выполняется арифметическое сложение множеств. Метод возвращает указатель на новый объект MultiSet, в конструктор которого передаются полученные в результате выполнения действия кратности.

```

1  unique_ptr<BaseSet> BaseSet::operator+(BaseSet& set) {
2      string set_name = this->name + " + " + set.name;
3
4      EmptySet* emptyset1 = dynamic_cast<EmptySet*>(this);
5      EmptySet* emptyset2 = dynamic_cast<EmptySet*>(&set);
6      if (emptyset1 && emptyset2) return
7          make_unique<EmptySet>(emptyset1->GetUniversum(), set_name);
8      else if (emptyset1) {
9          MultiSet* multiset1 = dynamic_cast<MultiSet*>(&set);
10         return make_unique<MultiSet>(*multiset1, set_name);
11     }
12     else if (emptyset2) {
13         MultiSet* multiset2 = dynamic_cast<MultiSet*>(this);
14         return make_unique<MultiSet>(*multiset2, set_name);
15     }
16
17     MultiSet* multiset1 = dynamic_cast<MultiSet*>(this);
18     MultiSet* multiset2 = dynamic_cast<MultiSet*>(&set);
19     vector<int> multiplicities;
20     for (int i = 0; i < pow(2, this->elements_size); i++) {
21         multiplicities.push_back(min(multiset1->main_vector[i] +
22             set.main_vector[i],
23             multiset1->GetUniversum().GetMainVectorCopy()[i]));
24     }
25
26     if (multiset1) {
27         return make_unique<MultiSet>(multiset1->GetUniversum(), set_name,
28             multiplicities);
29     }
30     if (multiset2) {
31         return make_unique<MultiSet>(multiset2->GetUniversum(), set_name,
32             multiplicities);
33     }
34 }

```

### 2.1.8 operator-

Вход: 2 мультимножества (А и В) и название нового мультимножества, которое будет получено после выполнения действия арифметической разности.

Выход: новое мультимножество - результат выполнения арифметической разности для двух переданных мультимножеств.

Метод проверяет, является ли какое-либо из переданных множеств пустым (объектом класса EmptySet). Если пустое множество А, то метод возвращает пустое множество (указатель на новый объект EmptySet). Если пустое множество В, то он возвращает множество А (указатель на объект MultiSet). Если же оба множества непустые, то результатом является новое мультимножество, кратности элементов которого получаются согласно правилу, по которому выполняется арифметическая разность мультимножеств. Результат выполнения действия может оказаться пустым множеством (в случае, если кратности всех элементов из первого множества соответственно меньше кратностей всех элементов из второго множества) и тогда метод вернет указатель на новый объект EmptySet. Иначе метод возвращает

указатель на новый объект MultiSet, в конструктор которого передаются полученные в результате выполнения действия кратности.

```

1  unique_ptr<BaseSet> BaseSet::operator-(BaseSet& set) {
2      string set_name = this->name + " - " + set.name;
3
4      EmptySet* emptyset1 = dynamic_cast<EmptySet*>(this);
5      EmptySet* emptyset2 = dynamic_cast<EmptySet*>(&set);
6      if (emptyset1) {
7          return make_unique<EmptySet>(emptyset1->GetUniversum(), set_name);
8      }
9      else if (emptyset2) {
10         MultiSet* multiset2 = dynamic_cast<MultiSet*>(this);
11         return make_unique<MultiSet>(*multiset2, set_name);
12     }
13
14     MultiSet* multiset1 = dynamic_cast<MultiSet*>(this);
15     MultiSet* multiset2 = dynamic_cast<MultiSet*>(&set);
16     bool zero_flag = true;
17     vector<int> multiplicities;
18     for (int i = 0; i < pow(2, this->elements_size); i++) {
19         multiplicities.push_back(max(multiset1->main_vector[i] -
20             set.main_vector[i], 0));
21         if (zero_flag && max(multiset1->main_vector[i] -
22             set.main_vector[i], 0) != 0) {
23             zero_flag = false;
24         }
25     }
26     if (multiset1) {
27         if (zero_flag) return
28             make_unique<EmptySet>(multiset1->GetUniversum(), set_name);
29         return make_unique<MultiSet>(multiset1->GetUniversum(), set_name,
30             multiplicities);
31     }
32     if (multiset2) {
33         if (zero_flag) return
34             make_unique<EmptySet>(multiset2->GetUniversum(), set_name);
35         return make_unique<MultiSet>(multiset2->GetUniversum(), set_name,
36             multiplicities);
37     }
38 }

```

### 2.1.9 operator\*

Вход: 2 мультимножества и название нового мультимножества, которое будет получено после выполнения действия арифметического произведения.

Выход: новое мультимножество - результат выполнения арифметического произведения для двух переданных мультимножеств.

Метод проверяет, является ли какое-либо из переданных множеств пустым (объектом класса EmptySet). Если хотя бы одно из множеств пустое, то метод возвращает пустое множество (указатель на новый объект EmptySet). Если же оба множества непустые, то результатом является новое мультимножество, кратности элементов которого получаются согласно правилу, по которому выполняется арифметическое произведение мультимножеств. Метод возвращает указатель на новый объект MultiSet, в конструктор которого передаются

полученные в результате выполнения действия кратности.

```

1  unique_ptr<BaseSet> BaseSet::operator*(BaseSet& set) {
2      string set_name = this->name + " * " + set.name;
3
4      EmptySet* emptyset1 = dynamic_cast<EmptySet*>(this);
5      EmptySet* emptyset2 = dynamic_cast<EmptySet*>(&set);
6      if (emptyset1 || emptyset2) {
7          if (emptyset1) {
8              return make_unique<EmptySet>(emptyset1->GetUniversum(),
9                  set_name);
10         }
11         return make_unique<EmptySet>(emptyset2->GetUniversum(), set_name);
12     }
13     MultiSet* multiset1 = dynamic_cast<MultiSet*>(this);
14     MultiSet* multiset2 = dynamic_cast<MultiSet*>(&set);
15     vector<int> multiplicities;
16     for (int i = 0; i < pow(2, this->elements_size); i++) {
17         multiplicities.push_back(min(multiset1->main_vector[i] *
18             set.main_vector[i],
19             multiset1->GetUniversum().GetMainVectorCopy()[i]));
20     }
21     if (multiset1) {
22         return make_unique<MultiSet>(multiset1->GetUniversum(), set_name,
23             multiplicities);
24     }
25     if (multiset2) {
26         return make_unique<MultiSet>(multiset2->GetUniversum(), set_name,
27             multiplicities);
28     }
29 }

```

### 2.1.10 operator/

Вход: 2 мультимножества (А и В) и название нового мультимножества, которое будет получено после выполнения действия арифметического деления.

Выход: новое мультимножество - результат выполнения арифметического деления для двух переданных мультимножеств.

Метод проверяет, является ли какое-либо из переданных множеств пустым (объектом класса EmptySet). Если хотя бы одно из множеств пустое, то метод возвращает пустое множество (указатель на новый объект EmptySet). Если же оба множества непустые, то результатом является новое мультимножество, кратности элементов которого получают-ся согласно правилу, по которому выполняется арифметическое деление мультимножеств. Результат выполнения действия может оказаться пустым множеством (в случае, если деления кратностей всех элементов из первого множества на соответствующие кратности всех элементов из второго множества равны нулю) и тогда метод вернет указатель на новый объект EmptySet. Иначе метод возвращает указатель на новый объект MultiSet, в конструктор которого передаются полученные в результате выполнения действия кратности.

```

1  unique_ptr<BaseSet> BaseSet::operator/(BaseSet& set) {
2      string set_name = this->name + " / " + set.name;
3
4      EmptySet* emptyset1 = dynamic_cast<EmptySet*>(this);

```

```

5     EmptySet* emptyset2 = dynamic_cast<EmptySet*>(&set);
6     if (emptyset1 || emptyset2) {
7         if (emptyset1) {
8             return make_unique<EmptySet>(emptyset1->GetUniversum(),
                set_name);
9         }
10        return make_unique<EmptySet>(emptyset2->GetUniversum(), set_name);
11    }
12
13    MultiSet* multiset1 = dynamic_cast<MultiSet*>(this);
14    MultiSet* multiset2 = dynamic_cast<MultiSet*>(&set);
15    bool zero_flag = true;
16    vector<int> multiplicities;
17    for (int i = 0; i < pow(2, this->elements_size); i++) {
18        if (set.main_vector[i] == 0) multiplicities.push_back(0);
19        else multiplicities.push_back(max(multiset1->main_vector[i] /
                set.main_vector[i], 0));
20        if (set.main_vector[i] != 0) {
21            if (zero_flag && max(multiset1->main_vector[i] /
                set.main_vector[i], 0) != 0) {
22                zero_flag = false;
23            }
24        }
25    }
26
27    if (multiset1) {
28        if (zero_flag) return
            make_unique<EmptySet>(multiset1->GetUniversum(), set_name);
29        return make_unique<MultiSet>(multiset1->GetUniversum(), set_name,
            multiplicities);
30    }
31    if (multiset2) {
32        if (zero_flag) return
            make_unique<EmptySet>(multiset2->GetUniversum(), set_name);
33        return make_unique<MultiSet>(multiset2->GetUniversum(), set_name,
            multiplicities);
34    }
35 }

```

### 2.1.11 operator«

Вход: любой экземпляр класса BaseSet.

Выход: вывод переданного мультимножества на экран.

Метод выводит переданное мультимножество на экран. Для универсума разрядности 2, например, вывод будет следующим:

Множество U: {00: 9757, 10: 2373, 11: 8694, 01: 30041}

Мощность множества U: 50865.

Кратности элементов метод берет из вектора main\_vector, а представления элементов получает с помощью бинарного кода Грея, используя метод GetGrayFunctionElement.

Если переданное мультимножество пустое, то метод выведет на экран следующий текст: «Множество U: Пустое множество!».

```

1     ostream& operator << (ostream & fout, BaseSet & set) {

```

```

2      EmptySet* emptyset1 = dynamic_cast<EmptySet*>(&set);
3      if (emptyset1) {
4          fout << "Множество " << set.name << ": Пустоемножество !" << endl;
5          return fout;
6      }
7
8      fout << "Множество " << set.name << ": {" ;
9
10     vector<int> element(set.elements_size);
11
12     int elements_count = pow(2, set.elements_size);
13     for (int i = 0; i < elements_count; i++) {
14         element = set.GrayFunctionElement(element, i);
15         PrintGrayFunctionElement(element);
16         fout << ": " << set.main_vector[i];
17         if (i != elements_count - 1) {
18             fout << ", ";
19         }
20     }
21
22     fout << "}" << endl;
23
24     fout << "Мощность множества " << set.name << ": " << set.power <<
25         endl;
26     return fout;
27 }

```

### 2.1.12 GetGrayFunctionElement

Вход: номер текущего подмножества и вектор, который хранит в себе предыдущее подмножество.

Выход: полученное по алгоритму Грея текущее подмножество.

Метод генерирует представление элементов определённого мультимножества. Получая на вход предыдущее представление и номер текущего, он с помощью метода GrayPodFunction определяет индекс элемента предыдущего представления, подлежащего изменению (0 становится 1, 1 становится 0). Метод возвращает измененное представление.

```

1      vector<int>& BaseSet::GetGrayFunctionElement(vector<int>& temp, int
2          i) const {
3          if (i != 0) {
4              int p = GrayPodFunction(i);
5              temp[p - 1] = 1 - temp[p - 1];
6          }
7          return temp;
8      }

```

### 2.1.13 GrayPodFunction

Вход: номер текущего подмножества.

Выход: номер изменяемого индекса текущего подмножества.

Метод вычисляет индекс, который нужно добавить или удалить в векторе предыдущего подмножества, основываясь на двоичном представлении номера текущего подмножества.



```

1  int BaseSet::GrayPodFunction(int i) const {
2      int q = 1, j = i;
3      while (j % 2 == 0) {
4          j /= 2;
5          q += 1;
6      }
7      return q;
8  }

```

## 2.2 Классы наследники BaseSet

Все классы, перечисленные в данном пункте, наследуются от BaseSet и представляют из себя реализацию таких мультимножеств как: универсум, пустое множество и обычное мультимножество. Отличаются данные классы только генерацией вектора `main_set`, хранящего массив кратностей данных мультимножеств.

EmptySet и MultiSet хранят в себе указатель на универсум. Нужно это для генерации кратностей (только в случае MultiSet) и для выполнения большинства действий.

Существуют данные классы для упрощения читаемости и масштабирования кода.

### 2.2.1 Конструктор класса Universum

Вход: разрядность элементов универсума.

Выход: создание экземпляра класса Universum и заполнение вектора `main_vector` его кратностями.

Конструктор заполняет вектор `main_vector` кратностями универсума с учетом заданной разрядности. Сами кратности генерируются случайно и значение каждой отдельной кратности лежит на отрезке от 0 до 32767 (ограничение языка программирования).

```

1  Universum::Universum(short elements_size, string set_name) :
    BaseSet(elements_size, set_name) {
2      this->name = set_name;
3      this->elements_size = elements_size;
4      this->power = 0;
5
6      for (int i = 0; i < pow(2, this->elements_size); i++) {
7          short element_multiplicity = rand();
8          this->main_vector.push_back(element_multiplicity);
9          this->power += element_multiplicity;
10     }
11 }

```

### 2.2.2 Конструктор класса EmptySet

Вход: текущий универсум программы.

Выход: создание экземпляра класса EmptySet.

Конструктор присваивает полю `Universum* universum` ссылку на переданный в конструктор универсум.

```

1  EmptySet::EmptySet(Universum& universum) : BaseSet(0, "None") {
2      this->universum = &universum;
3  }

```

### 2.2.3 Конструктор класса MultiSet №1

Вход: текущий универсум программы, а также мощность и количество ненулевых элементов, которые должны получиться у данного множества при генерации.

Выход: создание экземпляра класса MultiSet и заполнение вектора `main_vector` его кратностями.

Основная задача данного конструктора - случайным образом сгенерировать кратности мультимножества так, чтобы его мощность и количество ненулевых элементов были равны переданным значениям.

Можно генерировать кратности таким образом, что каждая новая случайная кратность получается из отрезка от нуля до значения мощности. После генерации одной кратности она вычитается из значения мощности и алгоритм повторяется до того момента, пока мощность не станет равной нулю. Недостатком такого метода является то, что при увеличении разрядности элементов или при уменьшении мощности множества, все больше и больше кратностей будут равны нулю, т.к. распределение случайно сгенерированных кратностей при данном алгоритме неравномерно.

Во избежание этой проблемы, кратности в данной работе генерируются следующим образом:

1. Пусть разрядность элементов -  $n$ . Тогда количество кратностей мультимножества -  $2^n$ . Генерируется  $2^n - 1$  случайных чисел, каждое из которых принадлежит отрезку от нуля до значения мощности. Все эти числа добавляются в массив, первый элемент которого всегда 0, а последний - мощность мультимножества, т.е. в полученном массиве будет  $2^n + 1$  чисел.

2. Данный массив сортируется по возрастанию чисел. Кратностями мультимножества же будут являться разности соседних чисел данного массива, т.к. сумма таких разностей будет равна мощности мультимножества.

Например, если разрядность элементов равна двум, то количество кратностей равно четырем. Т.е. из массива  $[0, 10, 20, 54, 55]$  будут получены следующие кратности:  $[10, 10, 34, 1]$ , причем  $10 + 10 + 34 + 1 = 55$ . При данном алгоритме генерации, распределение случайно-сгенерированных кратностей получается равномернее, чем при алгоритме, описанном до этого.

3. Контроль количества ненулевых элементов происходит во время генерации массива. Т.к. нулевая кратность может получиться только если в массиве окажется два одинаковых числа (их разность будет давать 0), то при достижении лимита ненулевых элементов, алгоритм не будет давать возможность добавить в массив число, равное какому-либо из уже присутствующих там чисел.
4. Недостатком данного алгоритма является невозможность при генерации контролировать то, что полученная кратность любого элемента может оказаться больше кратности этого элемента в универсуме. Тогда, если вычисленная кратность превышает допустимый предел, то она становится равной кратности элемента в универсуме, а разница прибавляется к переменной `credit`.
5. После прохождения всех элементов и генерации их кратностей, необходимо распределить значение полученной переменной `credit` между элементами, кратность которых не превышает кратности этих элементов в универсуме. Для этого случайным образом происходит обращение к таким элементам, и пока переменная `credit` не станет равной нулю, их кратности увеличиваются на случайные значения.

```

1  MultiSet::MultiSet(Universum& universum, string set_name, long long
    power, int not_null_elements) :
    BaseSet(universum.GetElementsSize(), set_name) {
2  this->universum = &universum;
3
4  int element_multiplicity = 0;
5  int elements_count = pow(2, this->elements_size);
6  vector<long long> temp_vector = {0, power};
7  unordered_map<long long, int> temp_map = { {0, 1}, {power, 1} };
8  int zero_count = 0;
9  int element;
10
11  for (int i = 0; i < elements_count - 1; i++) {
12      element = get_random_int(power);
13      int element_in_temp_count = temp_map[element];
14
15      if (element_in_temp_count) {
16          if (elements_count - not_null_elements != zero_count) zero_count
              += 1;
17          else {
18              while (temp_map[element]) {
19                  element = get_random_int(power);
20              }
21          }
22      }
23
24      temp_vector.push_back(element);
25      temp_map[element] += 1;
26  }
27
28  sort(temp_vector.begin(), temp_vector.end());
29
30  int credit = 0;
31  for (int i = 0; i < elements_count; i++) {
32      element_multiplicity = temp_vector[i + 1] - temp_vector[i];
33
34      if (element_multiplicity > universum[i]) {
35          credit += element_multiplicity - universum[i];
36          element_multiplicity = universum[i];
37      }
38
39      this->main_vector.push_back(element_multiplicity);
40      this->power += element_multiplicity;
41  }
42  ;
43  int element_ind = rand() % (elements_count);
44  while (credit != 0) {
45      if (universum[element_ind] - this->main_vector[element_ind] >=
          credit) {
46          this->main_vector[element_ind] += credit;
47          this->power += credit;
48          credit = 0;
49      }
50      else {
51          int temp = rand() % (universum[element_ind] -
              this->main_vector[element_ind] + 1);

```

```

52         this->main_vector[element_ind] += temp;
53         this->power += temp;
54         credit -= temp;
55     }
56     element_ind = (element_ind + 1) % elements_count;
57 }

```

#### 2.2.4 Конструктор класса MultiSet №2

Вход: текущий универсум программы, и вектор, содержащий в себе введенные пользователем кратности.

Выход: создание экземпляра класса MultiSet и заполнение вектора main\_vector его кратностями.

Конструктор заполняет вектор main\_vector кратностями, введенными пользователем вручную.

```

1  MultiSet::MultiSet(Universum& universum, string set_name, vector<int>&
    multiplicities) : BaseSet(universum.GetElementsSize(), set_name) {
2      this->universum = &universum;
3
4      for (int i = 0; i < pow(2, this->elements_size); i++) {
5          this->main_vector.push_back(multiplicities[i]);
6          this->power += multiplicities[i];
7      }
8  }

```

### 2.3 Класс Screen

Класс Screen отвечает за взаимодействие с пользователем через консольный интерфейс. Он управляет выводом данных на экран, обеспечивает ввод данных от пользователя и организует получение информации из других частей программы. Основными функциями класса являются отображение меню и обработка пользовательского ввода. Screen хранит в себе три указателя на объекты класса BaseSet: universum, setA и setB.

#### 2.3.1 Конструктор

Вход: ссылки на универсум и два множества.

Выход: создание экземпляра класса Screen.

```

1  Screen::Screen(Universum& universum, BaseSet& set1, BaseSet& set2) {
2      this->universum = &universum;
3      this->setA = &set1;
4      this->setB = &set2;
5  }

```

#### 2.3.2 ProgramStart

Вход: настройки программы.

Выход: создание универсума и двух мультимножеств.

Функция запрашивает у пользователя входные данные (разрядность универсума). Создает универсум и два множества (A и B) через GenerateMultiSetMenu. Инициализирует объект класса Screen для управления программой. Запускает интерфейс главного меню через PrintMainMenu.

```

1 void ProgramStart() {
2     settings::programShouldEnd = false;
3
4     cout << "Необходимо создать универсум ! Введите его разрядность ( 0 до
5         24): ";
6     string error_message = "\Вы ввели некорректную разрядность \Попробуйте
7         еще раз : ";
8     int n = ProgramStartInputs(0, 24, error_message,
9         buffer_sizes::n_int_size);
10
11     if (n == 0) {
12         Universum TempUniversum(n, "U");
13         EmptySet MainEmptySet(TempUniversum, "U");
14         EmptySet a(TempUniversum, "A");
15         EmptySet b(TempUniversum, "B");
16         Screen screen(MainEmptySet, a, b);
17         screen.PrintMainMenu();
18     }
19     else {
20         Universum MainUniversum(n, "U");
21
22         cout << endl << "Ваш универсум:" << endl;
23         cout << MainUniversum << endl;
24
25         MultiSet a;
26         GenerateMultisetMenu(a, MainUniversum, "A");
27         cout << endl << "Полученное множество:" << endl << a << endl;
28
29         MultiSet b;
30         GenerateMultisetMenu(b, MainUniversum, "B");
31         cout << endl << "Полученное множество:" << endl << b << endl;
32
33         Screen screen(MainUniversum, a, b);
34         screen.PrintMainMenu();
35     }
36 }

```

### 2.3.3 GenerateMultisetMenu

Вход: ссылки на объект заполняемого множества и на универсум, имя множества.

Выход: начало заполнения полученного множества вручную или случайными кратностями в зависимости от выбора пользователя.

Функция запрашивает от пользователя выбор режима генерации множества (случайный или ручной ввод). При выборе случайного ввода вызывается GenerateRandomMultiSet, иначе GenerateMultiSet.

```

1 void GenerateMultisetMenu(BaseSet& multiset, Universum& universum,
2     string Multiset_name) {
3     if (settings::programShouldEnd) return;
4     cout << "Необходимо задать множество " << Multiset_name << ".
5         Выберите режим генерации : " << endl;
6     cout << "1. Заполнить множество случайными кратностями . " << endl;
7     cout << "2. Заполнить множество вручную . " << endl;
8     cout << "Ввод: ";
9
10    bool multiplicity_choice_flag = true;

```

```

9      while (multipicity_choice_flag) {
10         char user_input = getchar();
11         bool one_symb_input_flag = InputClear();
12
13         if (one_symb_input_flag) {
14             void (*user_choice)(MultiSet&, Universum&, string) = nullptr;
15
16             for (int i = 0; i < sizeof(main_menu_choices) /
17                  sizeof(main_menu_choices[0]); i++) {
18                 if (user_input == multiset_generate_choises[i].symb) {
19                     user_choice = multiset_generate_choises[i].func;
20                     break;
21                 }
22             }
23             if (user_choice) {
24                 MultiSet* multiset1 = dynamic_cast<MultiSet*>(&multiset);
25                 user_choice(*multiset1, universum, Multiset_name);
26                 multipicity_choice_flag = false;
27             }
28             else cout << "Вы обращаетесь к несуществующему действию
29                    ,
30                    попробуйте еще раз !\n\n Ввод: ";
31         }
32     }
33     else cout << "Некорректный ввод! Попробуйте еще раз : ";
34 }
35 }

```

### 2.3.4 GenerateRandomMultiSet

Вход: ссылки на объект заполняемого множества и на универсум, имя множества.

Выход: инициализация объекта множества через конструктор, генерирующий случайные кратности.

Функция создает объект MultiSet с заданной мощностью и числом ненулевых кратностей, введенных пользователем.

```

1      void GenerateRandomMultiSet(MultiSet& multiset, Universum& universum,
2                                   string set_name) {
3          cout << endl << "Рандомная генерация множества !" << endl;
4          cout << "Введите мощность генерируемого множества от ( 0 до " <<
5               universum.GetPower() << " ) : ";
6          string error_message = "\Вы ввели некорректную разрядность \Попробуйте
7          еще раз : ";
8          int power = ProgramStartInputs(0, universum.GetPower(),
9               error_message, buffer_sizes::power_int_size);
10
11          cout << "Введите
12               обязательное количество ненулевых элементов генерируемого множества ";
13          cout << "от ( 0 до " << min(pow(2, universum.GetElementsSize()),
14               power) << " ) : ";
15          string error_message2 = "\Вы
16               ввели некорректное количество ненулевых элементов \Попробуйте
17               еще раз : ";
18          int not_null = ProgramStartInputs(0, pow(2,
19               universum.GetElementsSize()), error_message2,
20               buffer_sizes::power_int_size);
21
22          multiset = MultiSet(universum, set_name, power, not_null);
23      }

```

### 2.3.5 GenerateMultiSet

Вход: ссылки на объект заполняемого множества и на универсум, имя множества.

Выход: инициализация объекта множества через конструктор, задающий введенные пользователем кратности.

Пользователь вручную задает кратности для каждого элемента множества. Функция создает объект MultiSet с введенными кратностями.

```
1 void GenerateMultiSet(MultiSet& multiset, Universum& universum, string
   set_name) {
2     cout << endl << "Ручная генерация множества !" << endl;
3     vector<int> temp_vector;
4     string error_message = "\\Вы ввели некорректную кратность \\Попробуйте
   еще раз : ";
5
6     for (int i = 0; i < pow(2, universum.GetElementsSize()); i++) {
7         cout << "Введите " << i + 1 << " элемент ( 0 до " << universum[i]
           << "): ";
8         int power = ProgramStartInputs(0, universum.GetPower(),
           error_message, buffer_sizes::power_int_size);
9         temp_vector.push_back(power);
10    }
11
12    multiset = MultiSet(universum, set_name, temp_vector);
13 }
```

### 2.3.6 PrintMainMenu

Вход: универсум и два мультимножества.

Выход: вывод главного меню программы в консоль.

Метод выводит содержимое универсума, множества А и множества В. Отображает список доступных действий и обрабатывает пользовательский ввод для выбора действия.

```
1 void Screen::PrintMainMenu() {
2     if (settings::programShouldEnd) return;
3     Clear();
4     cout << "Главное меню!\n";
5     cout <<
6         "\n";
7     cout << "Текущие множества:\n";
8     cout << endl << *this->universum << endl << *this->setA << endl <<
9         *this->setB << endl;
10    cout <<
11        "\n";
12    cout << "[1] - Объединение множеств .\n";
13    cout << "[2] - Пересечение множеств .\n";
14    cout << "[3] - Разность множеств .\n";
15    cout << "[4] - Симметрическая разность множеств .\n";
16    cout << "[5] - Дополнение множеств .\n";
17    cout << "[6] - Арифметические операции над множествами .\n";
18    cout <<
19        "\n";
20    cout << "[7] - Перезапустить программу .\n";
21    cout << "[8] - Изменить множество А.\n";
22    cout << "[9] - Изменить множество В.\n";
23    cout <<
24        "\n";
25 }
```

```

20     cout << "[x] – Выход из программы .\n";
21     cout <<
22         "\n";
23     bool main_menu_choice_flag = true;
24     while (main_menu_choice_flag) {
25         char user_main_menu_input = getchar();
26         bool one_symb_input_flag = InputClear();
27
28         if (one_symb_input_flag) {
29             void (Screen:: * user_choice)() = nullptr;
30
31             for (int i = 0; i < sizeof(main_menu_choices) /
32                 sizeof(main_menu_choices[0]); i++) {
33                 if (user_main_menu_input == main_menu_choices[i].symb) {
34                     user_choice = main_menu_choices[i].func;
35                     break;
36                 }
37             }
38             if (user_choice) {
39                 if (user_main_menu_input == 'x' || user_main_menu_input == '7'
40                     || user_main_menu_input == '8' || user_main_menu_input == '9')
41                     (*this.*user_choice)();
42                 else PrintActionMenu(user_choice);
43
44                 main_menu_choice_flag = false;
45             }
46             else cout << "Вы обращаетесь к несуществующему действию
47                 ,
48                 попробуйте еще раз !\n\n Ввод: ";
49         }
50     }
51 }

```

### 2.3.7 PrintActionMenu

Вход: указатель на метод действия, выбранного пользователем.

Выход: вывод меню выбранного действия в консоль.

Метод выводит результат действия, выбранного пользователем, обращаясь к переданному методу данного действия, и возвращает пользователя в главное меню.

```

1     void Screen::PrintActionMenu(void (Screen:: *user_choice)()) {
2         if (settings::programShouldEnd) return;
3
4         Clear();
5         cout <<
6             "\n";
7
8         cout << *this->universum << endl << *this->setA << endl <<
9             *this->setB << endl;
10        (*this.*user_choice)();
11
12        cout << "\n Введите любой символ , чтобы вернуться в меню : ";

```



```

11     char user_answer = getchar();
12
13     PrintMainMenu();
14 }

```

### 2.3.8 Меню действий

В данную категорию относятся все методы, выводящие результаты действий в консоль.

Вход: универсум и два мультимножества.

Выход: вывод результата выбранного действия в консоль.

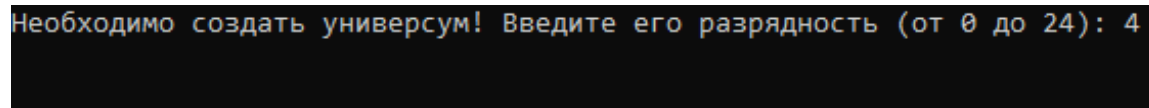
```

1  void Screen::PrintSetUnionMenu() {
2      cout << *SetUnion(*setA, *setB);
3  }
4
5  void Screen::PrintSetIntersectionMenu() {
6      cout << *SetIntersection(*setA, *setB);
7  }
8
9  void Screen::PrintDifferenceMenu() {
10     cout << *SetDifference(*setA, *setB) << endl;
11     cout << *SetDifference(*setB, *setA) << endl;
12 }
13
14 void Screen::PrintXorMenu() {
15     cout << *SetXor(*setA, *setB) << endl;
16 }
17
18 void Screen::PrintComplementMenu() {
19     EmptySet* emptyset = dynamic_cast<EmptySet*>(this->universum);
20     if (emptyset) {
21         Universum u(1, "");
22         EmptySet aa(u, "!A");
23         EmptySet bb(u, "!B");
24         cout << "U: " << aa << endl << "U: " << bb << endl;
25         return;
26     }
27
28     cout << *!setA << endl << *!setB << endl;
29 }
30
31 void Screen::PrintArithmeticMenu() {
32     cout <<>(*setA + *setB) << endl;
33     cout <<(*setA - *setB) << endl;
34     cout <<(*setA * *setB) << endl;
35     cout <<(*setA / *setB) << endl;
36     cout <<(*setB + *setA) << endl;
37     cout <<(*setB - *setA) << endl;
38     cout <<(*setB * *setA) << endl;
39     cout <<(*setB / *setA) << endl;
40 }

```

### 3 Результаты работы программы

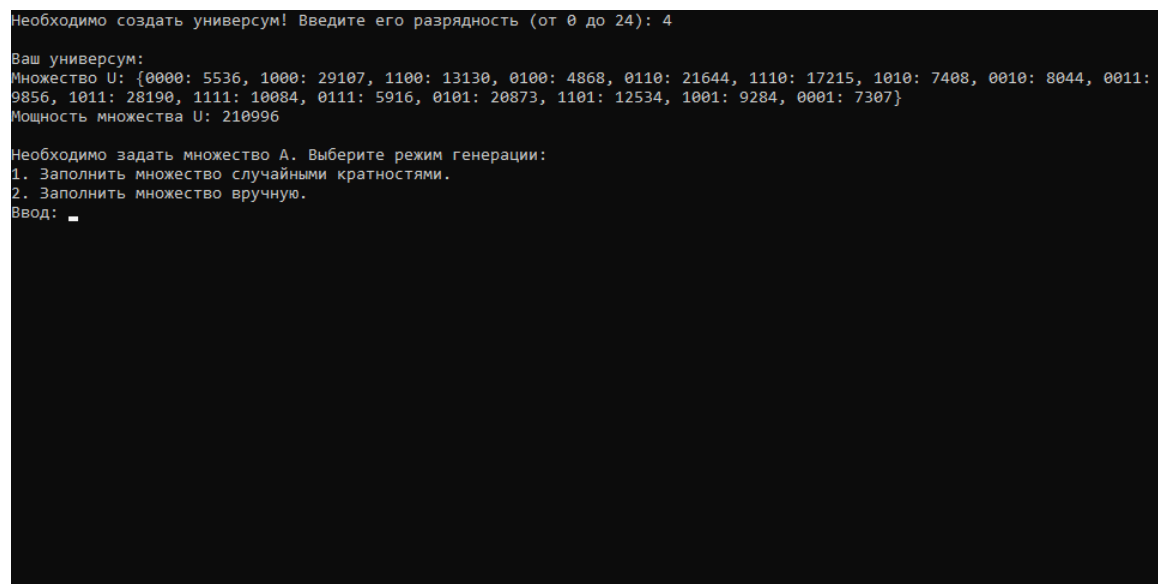
1) После запуска программы пользователя просят ввести разрядность универсума. (Рис.1).



```
Необходимо создать универсум! Введите его разрядность (от 0 до 24): 4
```

Рис. 1. Старт программы

2) После ввода разрядности, пользователю выводят получившийся универсум и просят задать множество A. (Рис.2).



```
Необходимо создать универсум! Введите его разрядность (от 0 до 24): 4
Ваш универсум:
Множество U: {0000: 5536, 1000: 29107, 1100: 13130, 0100: 4868, 0110: 21644, 1110: 17215, 1010: 7408, 0010: 8044, 0011: 9856, 1011: 28190, 1111: 10084, 0111: 5916, 0101: 20873, 1101: 12534, 1001: 9284, 0001: 7307}
Мощность множества U: 210996
Необходимо задать множество A. Выберите режим генерации:
1. Заполнить множество случайными кратностями.
2. Заполнить множество вручную.
Ввод: █
```

Рис. 2. Выбор способа генерации мультимножества A

3) При выборе заполнения множества случайными кратностями, пользователя попросят ввести мощность и количество ненулевых элементов создаваемого множества. (Рис.3).

```
Необходимо создать универсум! Введите его разрядность (от 0 до 24): 4

Ваш универсум:
Множество U: {0000: 5536, 1000: 29107, 1100: 13130, 0100: 4868, 0110: 21644, 1110: 17215, 1010: 7408, 0010: 8044, 0011: 9856, 1011: 28190, 1111: 10084, 0111: 5916, 0101: 20873, 1101: 12534, 1001: 9284, 0001: 7307}
Мощность множества U: 210996

Необходимо задать множество A. Выберите режим генерации:
1. Заполнить множество случайными кратностями.
2. Заполнить множество вручную.
Ввод: 1

Рандомная генерация множества!
Введите мощность генерируемого множества (от 0 до 210996): 55555
Введите обязательное количество ненулевых элементов генерируемого множества(от 0 до 16): 16_
```

Рис. 3. Ввод мощности и кол-ва ненулевых элементов

4) После генерации первого мультимножества, пользователь сможет увидеть его на экране и задать следующее множество. (Рис.4).

```
Необходимо создать универсум! Введите его разрядность (от 0 до 24): 4

Ваш универсум:
Множество U: {0000: 5536, 1000: 29107, 1100: 13130, 0100: 4868, 0110: 21644, 1110: 17215, 1010: 7408, 0010: 8044, 0011: 9856, 1011: 28190, 1111: 10084, 0111: 5916, 0101: 20873, 1101: 12534, 1001: 9284, 0001: 7307}
Мощность множества U: 210996

Необходимо задать множество A. Выберите режим генерации:
1. Заполнить множество случайными кратностями.
2. Заполнить множество вручную.
Ввод: 1

Рандомная генерация множества!
Введите мощность генерируемого множества (от 0 до 210996): 55555
Введите обязательное количество ненулевых элементов генерируемого множества(от 0 до 16): 16

Полученное множество:
Множество A: {0000: 5126, 1000: 4460, 1100: 396, 0100: 1539, 0110: 1233, 1110: 455, 1010: 7408, 0010: 7009, 0011: 2104, 1011: 1366, 1111: 6162, 0111: 4461, 0101: 2726, 1101: 1434, 1001: 9284, 0001: 392}
Мощность множества A: 55555

Необходимо задать множество B. Выберите режим генерации:
1. Заполнить множество случайными кратностями.
2. Заполнить множество вручную.
Ввод: 2_
```

Рис. 4. Выбор способа генерации мультимножества B

5) Если ввести число, выходящее за пределы, пользователь получит ошибку ввода. (Рис.5).

```
Полученное множество:
Множество A: {0000: 5126, 1000: 4460, 1100: 396, 0100: 1539, 0110: 1233, 1110: 455, 1010: 7408, 0010: 7009, 0011: 2104,
1011: 1366, 1111: 6162, 0111: 4461, 0101: 2726, 1101: 1434, 1001: 9284, 0001: 392}
Мощность множества A: 55555

Необходимо задать множество B. Выберите режим генерации:
1. Заполнить множество случайными кратностями.
2. Заполнить множество вручную.
Ввод: 2

Ручная генерация множества!
Введите 1 элемент (от 0 до 5536): 1
Введите 2 элемент (от 0 до 29107): 0
Введите 3 элемент (от 0 до 13130): 1
Введите 4 элемент (от 0 до 4868): 0
Введите 5 элемент (от 0 до 21644): 1
Введите 6 элемент (от 0 до 17215): 0
Введите 7 элемент (от 0 до 7408): 1
Введите 8 элемент (от 0 до 8044): 100000
Введите 9 элемент (от 0 до 9856): 1000000000000000
Некорректный ввод! Попробуйте еще раз:
```

Рис. 5. Обработка некорректного ввода

6) При выборе заполнения множества вручную, пользователя попросят ввести кратности элементов создаваемого множества. (Рис.6).

```
Полученное множество:
Множество A: {0000: 5126, 1000: 4460, 1100: 396, 0100: 1539, 0110: 1233, 1110: 455, 1010: 7408, 0010: 7009, 0011: 2104,
1011: 1366, 1111: 6162, 0111: 4461, 0101: 2726, 1101: 1434, 1001: 9284, 0001: 392}
Мощность множества A: 55555

Необходимо задать множество B. Выберите режим генерации:
1. Заполнить множество случайными кратностями.
2. Заполнить множество вручную.
Ввод: 2

Ручная генерация множества!
Введите 1 элемент (от 0 до 5536): 1
Введите 2 элемент (от 0 до 29107): 0
Введите 3 элемент (от 0 до 13130): 1
Введите 4 элемент (от 0 до 4868): 0
Введите 5 элемент (от 0 до 21644): 1
Введите 6 элемент (от 0 до 17215): 0
Введите 7 элемент (от 0 до 7408): 1
Введите 8 элемент (от 0 до 8044): 100000
Введите 9 элемент (от 0 до 9856): 1000000000000000
Некорректный ввод! Попробуйте еще раз: 5454
Введите 10 элемент (от 0 до 28190): 4
Введите 11 элемент (от 0 до 10084): 3
Введите 12 элемент (от 0 до 5916): 2
Введите 13 элемент (от 0 до 20873): 1
Введите 14 элемент (от 0 до 12534): 4
Введите 15 элемент (от 0 до 9284): 3
Введите 16 элемент (от 0 до 7307): 10_
```

Рис. 6. Ручной ввод кратностей множества

7) После успешной генерации множеств, пользователь окажется в главном меню программы. (Рис.7).

```

Текущие множества:

Множество U: {0000: 5536, 1000: 29107, 1100: 13130, 0100: 4868, 0110: 21644, 1110: 17215, 1010: 7408, 0010: 8044, 0011: 9856, 1011: 28190, 1111: 10084, 0111: 5916, 0101: 20873, 1101: 12534, 1001: 9284, 0001: 7307}
Мощность множества U: 210996

Множество A: {0000: 5126, 1000: 4460, 1100: 396, 0100: 1539, 0110: 1233, 1110: 455, 1010: 7408, 0010: 7009, 0011: 2104, 1011: 1366, 1111: 6162, 0111: 4461, 0101: 2726, 1101: 1434, 1001: 9284, 0001: 392}
Мощность множества A: 55555

Множество B: {0000: 1, 1000: 0, 1100: 1, 0100: 0, 0110: 1, 1110: 0, 1010: 1, 0010: 100000, 0011: 5454, 1011: 4, 1111: 3, 0111: 2, 0101: 1, 1101: 4, 1001: 3, 0001: 10}
Мощность множества B: 105485

-----
[1] - Объединение множеств.
[2] - Пересечение множеств.
[3] - Разности множеств.
[4] - Симметрическая разность множеств.
[5] - Дополнения множеств.
[6] - Арифметические операции над множествами.
-----
[7] - Перезапустить программу.
[8] - Изменить множество A.
[9] - Изменить множество B.
-----
[x] - Выход из программы.
-----

Ввод:

```

Рис. 7. Главное меню

8) При выборе первого действия, пользователю будет выведен результат объединения заданных множеств. (Рис.8).

```

-----
Множество U: {0000: 5536, 1000: 29107, 1100: 13130, 0100: 4868, 0110: 21644, 1110: 17215, 1010: 7408, 0010: 8044, 0011: 9856, 1011: 28190, 1111: 10084, 0111: 5916, 0101: 20873, 1101: 12534, 1001: 9284, 0001: 7307}
Мощность множества U: 210996

Множество A: {0000: 5126, 1000: 4460, 1100: 396, 0100: 1539, 0110: 1233, 1110: 455, 1010: 7408, 0010: 7009, 0011: 2104, 1011: 1366, 1111: 6162, 0111: 4461, 0101: 2726, 1101: 1434, 1001: 9284, 0001: 392}
Мощность множества A: 55555

Множество B: {0000: 1, 1000: 0, 1100: 1, 0100: 0, 0110: 1, 1110: 0, 1010: 1, 0010: 100000, 0011: 5454, 1011: 4, 1111: 3, 0111: 2, 0101: 1, 1101: 4, 1001: 3, 0001: 10}
Мощность множества B: 105485

Множество A \ B: {0000: 5126, 1000: 4460, 1100: 396, 0100: 1539, 0110: 1233, 1110: 455, 1010: 7408, 0010: 100000, 0011: 5454, 1011: 1366, 1111: 6162, 0111: 4461, 0101: 2726, 1101: 1434, 1001: 9284, 0001: 392}
Мощность множества A \ B: 151896

Введите любой символ, чтобы вернуться в меню : _

```

Рис. 8. Объединение множеств

9) При выборе второго действия, пользователю будет выведен результат пересечения заданных множеств. (Рис.9).

```

-----
Множество U: {0000: 5536, 1000: 29107, 1100: 13130, 0100: 4868, 0110: 21644, 1110: 17215, 1010: 7408, 0010: 8044, 0011: 9856, 1011: 28190, 1111: 10084, 0111: 5916, 0101: 20873, 1101: 12534, 1001: 9284, 0001: 7307}
Мощность множества U: 210996

Множество A: {0000: 5126, 1000: 4460, 1100: 396, 0100: 1539, 0110: 1233, 1110: 455, 1010: 7408, 0010: 7009, 0011: 2104, 1011: 1366, 1111: 6162, 0111: 4461, 0101: 2726, 1101: 1434, 1001: 9284, 0001: 392}
Мощность множества A: 55555

Множество B: {0000: 1, 1000: 0, 1100: 1, 0100: 0, 0110: 1, 1110: 0, 1010: 1, 0010: 100000, 0011: 5454, 1011: 4, 1111: 3, 0111: 2, 0101: 1, 1101: 4, 1001: 3, 0001: 10}
Мощность множества B: 105485

Множество A /\ B: {0000: 1, 1000: 0, 1100: 1, 0100: 0, 0110: 1, 1110: 0, 1010: 1, 0010: 7009, 0011: 2104, 1011: 4, 1111: 3, 0111: 2, 0101: 1, 1101: 4, 1001: 3, 0001: 10}
Мощность множества A /\ B: 9144

Введите любой символ, чтобы вернуться в меню : █

```

Рис. 9. Пересечение множеств

10) При выборе третьего действия, пользователю будет выведен результат разности заданных множеств. (Рис.10).

```

-----
Множество U: {0000: 5536, 1000: 29107, 1100: 13130, 0100: 4868, 0110: 21644, 1110: 17215, 1010: 7408, 0010: 8044, 0011: 9856, 1011: 28190, 1111: 10084, 0111: 5916, 0101: 20873, 1101: 12534, 1001: 9284, 0001: 7307}
Мощность множества U: 210996

Множество A: {0000: 5126, 1000: 4460, 1100: 396, 0100: 1539, 0110: 1233, 1110: 455, 1010: 7408, 0010: 7009, 0011: 2104, 1011: 1366, 1111: 6162, 0111: 4461, 0101: 2726, 1101: 1434, 1001: 9284, 0001: 392}
Мощность множества A: 55555

Множество B: {0000: 1, 1000: 0, 1100: 1, 0100: 0, 0110: 1, 1110: 0, 1010: 1, 0010: 100000, 0011: 5454, 1011: 4, 1111: 3, 0111: 2, 0101: 1, 1101: 4, 1001: 3, 0001: 10}
Мощность множества B: 105485

Множество A \ B: {0000: 5126, 1000: 4460, 1100: 396, 0100: 1539, 0110: 1233, 1110: 455, 1010: 7407, 0010: -91956, 0011: 2104, 1011: 1366, 1111: 6162, 0111: 4461, 0101: 2726, 1101: 1434, 1001: 9281, 0001: 392}
Мощность множества A \ B: -43414

Множество B \ A: {0000: 1, 1000: 0, 1100: 1, 0100: 0, 0110: 1, 1110: 0, 1010: 0, 0010: 1035, 0011: 5454, 1011: 4, 1111: 3, 0111: 2, 0101: 1, 1101: 4, 1001: 0, 0001: 10}
Мощность множества B \ A: 6516

Введите любой символ, чтобы вернуться в меню : █

```

Рис. 10. Разность множеств

11) При выборе четвертого действия, пользователю будет выведен результат симметрической разности заданных множеств. (Рис.11).

```

-----
Множество U: {0000: 5536, 1000: 29107, 1100: 13130, 0100: 4868, 0110: 21644, 1110: 17215, 1010: 7408, 0010: 8044, 0011: 9856, 1011: 28190, 1111: 10084, 0111: 5916, 0101: 20873, 1101: 12534, 1001: 9284, 0001: 7307}
Мощность множества U: 210996

Множество A: {0000: 5126, 1000: 4460, 1100: 396, 0100: 1539, 0110: 1233, 1110: 455, 1010: 7408, 0010: 7009, 0011: 2104, 1011: 1366, 1111: 6162, 0111: 4461, 0101: 2726, 1101: 1434, 1001: 9284, 0001: 392}
Мощность множества A: 55555

Множество B: {0000: 1, 1000: 0, 1100: 1, 0100: 0, 0110: 1, 1110: 0, 1010: 1, 0010: 100000, 0011: 5454, 1011: 4, 1111: 3, 0111: 2, 0101: 1, 1101: 4, 1001: 3, 0001: 10}
Мощность множества B: 105485

Множество A |+| B: {0000: 5126, 1000: 4460, 1100: 396, 0100: 1539, 0110: 1233, 1110: 455, 1010: 7407, 0010: 1035, 0011: 5454, 1011: 1366, 1111: 6162, 0111: 4461, 0101: 2726, 1101: 1434, 1001: 9281, 0001: 392}
Мощность множества A |+| B: 52927

Введите любой символ, чтобы вернуться в меню :

```

Рис. 11. Симметрическая разность множеств

12) При выборе пятого действия, пользователю будет выведен результат дополнения заданных множеств. (Рис.12).

```

-----
Множество U: {0000: 5536, 1000: 29107, 1100: 13130, 0100: 4868, 0110: 21644, 1110: 17215, 1010: 7408, 0010: 8044, 0011: 9856, 1011: 28190, 1111: 10084, 0111: 5916, 0101: 20873, 1101: 12534, 1001: 9284, 0001: 7307}
Мощность множества U: 210996

Множество A: {0000: 5126, 1000: 4460, 1100: 396, 0100: 1539, 0110: 1233, 1110: 455, 1010: 7408, 0010: 7009, 0011: 2104, 1011: 1366, 1111: 6162, 0111: 4461, 0101: 2726, 1101: 1434, 1001: 9284, 0001: 392}
Мощность множества A: 55555

Множество B: {0000: 1, 1000: 0, 1100: 1, 0100: 0, 0110: 1, 1110: 0, 1010: 1, 0010: 100000, 0011: 5454, 1011: 4, 1111: 3, 0111: 2, 0101: 1, 1101: 4, 1001: 3, 0001: 10}
Мощность множества B: 105485

Множество !A: {0000: 410, 1000: 24647, 1100: 12734, 0100: 3329, 0110: 20411, 1110: 16760, 1010: 0, 0010: 1035, 0011: 7752, 1011: 26824, 1111: 3922, 0111: 1455, 0101: 18147, 1101: 11100, 1001: 0, 0001: 6915}
Мощность множества !A: 155441

Множество !B: {0000: 5535, 1000: 29107, 1100: 13129, 0100: 4868, 0110: 21643, 1110: 17215, 1010: 7407, 0010: -91956, 0011: 4402, 1011: 28186, 1111: 10081, 0111: 5914, 0101: 20872, 1101: 12530, 1001: 9281, 0001: 7297}
Мощность множества !B: 105511

Введите любой символ, чтобы вернуться в меню : _

```

Рис. 12. Дополнение множеств

13) При выборе шестого действия, пользователю будет выведен результат арифметических операций над заданными множествами. (Рис.13).

```

Множество A - B: {0000: 5125, 1000: 4460, 1100: 395, 0100: 1539, 0110: 1232, 1110: 455, 1010: 7407, 0010: 0, 0011: 0, 1011: 1362, 1111: 6159, 0111: 4459, 0101: 2725, 1101: 1430, 1001: 9281, 0001: 382}
Мощность множества A - B: 46411

Множество A * B: {0000: 5126, 1000: 0, 1100: 396, 0100: 0, 0110: 1233, 1110: 0, 1010: 7408, 0010: 8044, 0011: 9856, 1011: 5464, 1111: 10084, 0111: 5916, 0101: 2726, 1101: 5736, 1001: 9284, 0001: 3920}
Мощность множества A * B: 75193

Множество A / B: {0000: 5126, 1000: 0, 1100: 396, 0100: 0, 0110: 1233, 1110: 0, 1010: 7408, 0010: 0, 0011: 0, 1011: 341, 1111: 2054, 0111: 2230, 0101: 2726, 1101: 358, 1001: 3094, 0001: 39}
Мощность множества A / B: 25005

Множество B + A: {0000: 5127, 1000: 4460, 1100: 397, 0100: 1539, 0110: 1234, 1110: 455, 1010: 7408, 0010: 8044, 0011: 7558, 1011: 1370, 1111: 6165, 0111: 4463, 0101: 2727, 1101: 1438, 1001: 9284, 0001: 402}
Мощность множества B + A: 62071

Множество B - A: {0000: 0, 1000: 0, 1100: 0, 0100: 0, 0110: 0, 1110: 0, 1010: 0, 0010: 92991, 0011: 3350, 1011: 0, 1111: 0, 0111: 0, 0101: 0, 1101: 0, 1001: 0, 0001: 0}
Мощность множества B - A: 96341

Множество B * A: {0000: 5126, 1000: 0, 1100: 396, 0100: 0, 0110: 1233, 1110: 0, 1010: 7408, 0010: 8044, 0011: 9856, 1011: 5464, 1111: 10084, 0111: 5916, 0101: 2726, 1101: 5736, 1001: 9284, 0001: 3920}
Мощность множества B * A: 75193

Множество B / A: {0000: 0, 1000: 0, 1100: 0, 0100: 0, 0110: 0, 1110: 0, 1010: 0, 0010: 14, 0011: 2, 1011: 0, 1111: 0, 0111: 0, 0101: 0, 1101: 0, 1001: 0, 0001: 0}
Мощность множества B / A: 16

Введите любой символ, чтобы вернуться в меню :

```

Рис. 13. Арифметические операции над множествами

14) При выборе восьмого и девятого действий, пользователь сможет перезадавать множества. (Рис.14).

```

Необходимо задать множество A. Выберите режим генерации:
1. Заполнить множество случайными кратностями.
2. Заполнить множество вручную.
Ввод: 1

Рандомная генерация множества!
Введите мощность генерируемого множества (от 0 до 210996):

```

Рис. 14. Арифметические операции над множествами

15) Обработка некорректных вводов. (Рис.15).



```
Необходимо задать множество A. Выберите режим генерации:  
1. Заполнить множество случайными кратностями.  
2. Заполнить множество вручную.  
Ввод: 1  
  
Рандомная генерация множества!  
Введите мощность генерируемого множества (от 0 до 210996): 3434342  
  
Вы ввели некорректную разрядность  
Попробуйте еще раз: 43  
Введите обязательное количество ненулевых элементов генерируемого множества(от 0 до 16): fdfs  
Некорректный ввод! Попробуйте еще раз: s55252  
Некорректный ввод! Попробуйте еще раз: sdfsd352  
Некорректный ввод! Попробуйте еще раз: dsfsdf342  
Некорректный ввод! Попробуйте еще раз: fsdtm23kn34h2r23fd3  
Некорректный ввод! Попробуйте еще раз: 4747  
  
Вы ввели некорректное количество ненулевых элементов  
Попробуйте еще раз:
```

Рис. 15. Некорректный ввод

## Заключение

В результате выполнения данной лабораторной работы был реализован алгоритм генерации бинарного кода Грея, на основе которого были созданы универсальное множество (универсум), а также два мультимножества A и B. Разрядность всех множеств определяется числом, введенным пользователем. После введения разрядности, заполнение множеств A и B может производиться двумя способами: автоматически и вручную, способ выбирает пользователь. При выборе автоматического способа генерации множеств, пользователю требуется ввести мощность и количество ненулевых элементов генерируемого множества.

В работе реализовано множество функций, позволяющих получить результаты следующих действий над множествами: объединение, пересечение, разность (оба варианта), симметрическая разность, дополнение (оба варианта), а также арифметические сумма, разность, произведение и деление

Также программа контролирует пользовательский ввод и не дает пользователю ввести некорректные данные.

Достоинства программы:

- Использование готовых решений STL контейнеров дает быстрое действие работы программы и дополнительную защиту от утечек памяти.
- Реализация различных видов множеств в отдельных классах дает возможность легкого и быстрого использования результатов работы при масштабировании программы.
- Усовершенствованный в ходе работы алгоритм равномерного распределения случайных чисел позволяет улучшить опыт взаимодействия с программой, т.к. благодаря данному алгоритму, множества, получаемые случайным образом, при каждой новой генерации будут сильно отличаться, что положительно повлияет на результаты исследований и работу с программой.
- Максимальная разрядность множеств программы равна 24. При такой большой разрядности, для большинства комбинаций мощности и количества ненулевых элементов, множества будут генерироваться в течение 10 секунд - минуты. Множества с меньшими разрядностями обычно генерируются моментально.

Недостатки программы:

- В главной части алгоритма генерации множеств невозможно контролировать то, что полученная кратность любого элемента может оказаться больше кратности этого элемента в универсуме.
- Часть алгоритма с использованием переменной `credit` и реализация контроля количества ненулевых элементов обладают достаточно большой итеративностью, что может повлиять на скорость работы программы в некоторых случаях.
- Код потенциально сложен для поддержки, т.к. логика корректировки значений, полученных алгоритмом генерации случайных мультимножеств, может быть трудной для понимания, что увеличивает трудозатраты на масштабирование программы.
- Нет возможности возвращаться на предыдущий этап выполнения программы.

Масштабирование: в программу можно добавить следующие функции:

- Добавление и работа по выбору с тремя различными мультимножествами.

- Возможность вводить формулы, содержащие действия над мультимножествами, используемые в работе (объединение, пересечение, разность симметрическая разность, дополнение). Программа должна уметь интерпретировать их для вычисления результата.

## Список литературы

1. Секция "Телематика"/ текст : электронный / — URL: <https://tema.spbstu.ru/dismath/> (Дата обращения 14.01.2025).
2. Новиков Ф. А. Дискретная математика для программистов. 3-е изд. — Санкт-Петербург: Питер Пресс, 2009. - 384стр.
3. Microsoft C++ Standart Library Documentation / текст : электронный / - URL: <https://learn.microsoft.com/ru-ru/cpp/standard-library/> (Дата обращения 14.01.2025).