БУ ВО Ханты-Мансийского автономного округа – Югры

«Сургутский государственный университет»

Политехнический институт

Кафедра автоматики и компьютерных систем

**Пояснительная записка**

К курсовому проекту

по дисциплине «Структуры и алгоритмы обработки данных»

Выполнил: Бельтюков М. О.

студент группы 609-22

Проверил: Старший преподаватель кафедры автоматики и компьютерных систем Назаров Е.В.

Сургут

2024 г.

# Задание

Вариант 14 – Класс «Множество».

В соответствии с выбранным вариантом необходимо:

1. Провести анализ предметной области возможного применения проектируемого класса.

2. Провести анализ функциональности проектируемого класса.

3. Разработать интерфейс класса.

4. В соответствии с разработанным интерфейсом спроектировать тестовое приложение.

5. Выполнить проектирование класса, обоснованно выбирая:

- необходимые поля класса;

- методы класса, включив в обязательном порядке операции вставки, удаления, изменения, поиска отдельных элементов, входящих в класс.

6. Провести проектирование алгоритмов, лежащих в основе разрабатываемых методов.

7. Реализовать полученное проектное решение.

8. Реализовать тестовое приложение и провести тестирование разработанного и реализованного класса.

9. Провести исследование одной из операций (вставка, удаление, изменение, поиск) над элементами. Построить зависимость времени выполнения операции от числа элементов, над которыми она выполняется.

10. Оценить асимптотическую сложность реализованных алгоритмов вставки, удаления и поиска элементов класса.

# Аннотация

Курсовой проект на тему: Класс «Множество».

Выполнил - Бельтюков М.О.

Руководитель - Назаров Е.В.

Пояснительная записка включает следующие разделы – содержание, введение, проектирование, кодирование, тестирование и исследование операции вставки во множество.

Цель данной работы – разработка класса, осуществляющего работу структуры данных “Множество”.

В результате работы было разработано приложение, работающие с классом “множество” и класс множество.

# Содержание

[Введение 6](#_Toc169165654)

[1. Анализ 7](#_Toc169165655)

[1.1 Цель и задачи 7](#_Toc169165656)

[1.2 Анализ предметной области 7](#_Toc169165657)

[1.3 Асимптотическая сложность множества 8](#_Toc169165658)

[1.4 Обзор аналогов 8](#_Toc169165659)

[1.4.1 Аналог 1 – Неупорядоченное множество из STL, С++ 8](#_Toc169165660)

[1.4.2 Аналог 2 – Неупорядоченное множество из python3 9](#_Toc169165661)

[1.4.3 Аналог 3 – Неупорядоченное множество из java8 10](#_Toc169165662)

[1.3 Обобщение частной задачи 11](#_Toc169165663)

[2. Проектирование 12](#_Toc169165664)

[2.1 Формальное описание задачи 12](#_Toc169165665)

[2.2 Основные алгоритмы 12](#_Toc169165666)

[2.2.1 Вставка элемента 12](#_Toc169165667)

[2.2.2 Удаление узла 13](#_Toc169165668)

[2.2.3 Вывод множества 13](#_Toc169165669)

[2.2.4 Проверка наличия элемента в множестве 13](#_Toc169165670)

[2.2.5 Копирование множеств 14](#_Toc169165671)

[2.2.6 Объединение множеств 14](#_Toc169165672)

[2.2.7 Пересечение множеств 14](#_Toc169165673)

[2.2.8 Разность множеств 15](#_Toc169165674)

[2.2.9 Сравнение множеств 15](#_Toc169165675)

[2.2.10 Отчистка множества 15](#_Toc169165676)

[2.2.11 Проверка множества на пустоту 15](#_Toc169165677)

[3. Кодирование 16](#_Toc169165678)

[3.1 Представление данных в памяти при их обработке 16](#_Toc169165679)

[3.2 Структура файлов 18](#_Toc169165680)

[3.3 Методы класса узел 20](#_Toc169165681)

[3.4 Методы класса множество 21](#_Toc169165682)

[3.4.1 Приватные методы класса множество 21](#_Toc169165683)

[3.4.2 Защищённые методы класса множество 22](#_Toc169165684)

[3.4.3 Публичные методы класса множество 23](#_Toc169165685)

[3.5 Разработка кода сборочного файла 27](#_Toc169165686)

[4. Тестирование 29](#_Toc169165687)

[4.1 Поведенческая модель программы 29](#_Toc169165688)

[4.2 Проверка работоспособности класса 30](#_Toc169165689)

[5. Исследование операции вставки во множество 34](#_Toc169165690)

[Заключение 40](#_Toc169165691)

[Список использованных источников 41](#_Toc169165692)

[Приложения 43](#_Toc169165693)

[Приложения А. Рисунки 43](#_Toc169165694)

[Приложения Б. Листинги 46](#_Toc169165695)

# Введение

Множество — это совокупность уникальных элементов, это математическая концепция, реализованная во многих языках программирования. В информатике заданная структура данных — это структура данных, которая хранит коллекцию уникальных элементов и обеспечивает эффективные операции по добавлению, удалению и запросу на наличие элементов. Возможное применение при работе с большими наборами данных:

* Отфильтровать повторяющиеся элементы.
* Проверить, находится ли элемент в множестве.

Преимущество использования множества заключается, например, в том, что данная структура данных имеет константное время сложности выполнения этих операций. [1]

# 1. Анализ

## 1.1 Цель и задачи

Необходимо реализовать программу, благодаря которой можно будет использовать структуру данных “множество”. Таким образом можно выделить следующие задачи:

1. Определить тип множества – упорядоченное или неупорядоченное.
2. Реализовать вставку элемента в множество.
3. Реализовать удаление элемента из множества.
4. Реализовать вывод множества.
5. Реализовать распространённые функции для работы с множеством, набор которых будет определён после пункта “анализ”.

## 1.2 Анализ предметной области

Множество может быть реализовано различными способами, но наиболее распространёнными являются:

* Множество на основе хэша – представлено в виде хэш-таблицы, где каждый элемент хранится в сегменте на основе хэш-кода.
* Множество на основе структуры данных “деревья” – В этой реализации набор представлен как двоичное дерево поиска, где каждый узел представляет собой элемент набора. [2]

Упорядоченное множество (множество на основе деревьев поиска) используется, когда:

* Необходимо выводить отсортированные данные.
* Необходимо использовать функции, которым необходимы отсортированные данные, например, бинарный поиск.

Неупорядоченное множество (множество на основе хэш-таблицы) используется, когда:

* Упорядоченность не имеет смысла.
* Нет необходимости в обходе. [3]

## 1.3 Асимптотическая сложность множества

Асимптотическая сложность упорядоченного и неупорядоченного множества представлена на таблице 1.

Таблица 1 - Асимптотическая сложность упорядоченного и неупорядоченного множества

|  |  |  |  |
| --- | --- | --- | --- |
| Множество | Добавление элемента | Удаление элемента | Проверка наличия элемента |
| Упорядоченное | O(log(n)) | O(log(n)) | O(log(n)) |
| Неупорядоченное | O(1) | O(1) | O(1) |

В связи с тем, что асимптотическая сложность неупорядоченного множества лучше, чем у упорядоченного, было решено реализовать неупорядоченное множество.

Далее будет рассмотрено 3 реализации из неупорядоченного множества из разных языков программирования, чтобы определится с базовыми функциями множества.

## 1.4 Обзор аналогов

### 1.4.1 Аналог 1 – Неупорядоченное множество из STL, С++

Особенности реализации неупорядоченного множества в STL:

Для хранения элементов, тип которых определяется разработчиком, необходимо указать:

* Тип хранимого значения.
* Хеш-функцию, которая принимает элемент хранимого типа и возвращает на его основе хеш-код.
* Бинарный предикат, принимающий 2 аргумента того же типа, что и элементы и возвращающий истину в случае, когда элементы равны и ложь в случае, когда элементы не являются равными.
* Тип объекта распредилителя, используемый для определения модели распределения памяти.

Полный список открытых методов представлен в приложении А.1. Некоторые из них выделены в таблице 2 [4]:

Таблица 2 – неполный список методов класса неупорядоченное множество в STL, c++

|  |  |
| --- | --- |
| Метод | Описание |
| insert | Вставка элемента в неупорядоченное множество |
| Size | Получение количества элементов в неупорядоченном множестве |
| clear | очищает содержимое неупорядоченного множества |
| erase | Удаляет либо один элемент, либо диапазон элементов из неупорядоченного множества |
| find | Ищет элемент с определённым ключом в неупорядоченном множестве |
| operator== | Сравнивает значения в неупорядоченном множестве |

### 1.4.2 Аналог 2 – Неупорядоченное множество из python3

Особенности реализации неупорядоченного множества в python3:

Для хранения элементов, тип которых определяется разработчиком, необходимо в реализуемом типе данных переопределить следующие методы:

1. Метод “\_\_eq\_\_” – метод для сравнения двух элементов хранимого типа.
2. Метод “\_\_hash\_\_” – метод для получения хеш-кода для хранимого типа.

Полный список методов представлен в приложениях А.2.1 и А.2.2. В таблице 3 были выделены некоторые из них [5]:

Таблица 3 - неполный список методов класса неупорядоченное множество в python3

|  |  |
| --- | --- |
| Метод | Описание |
| |= | Добавление элементов в неупорядоченное множество |
| &= | Сохраняет только совпадающие элементы из неупорядоченных множеств |
| -= | Удалит элементы, найденные в других неупорядоченных множествах |
| ^= | Исключает элементы, которые встречаются в обоих неупорядоченных множествах |
| clear | Очищает содержимое неупорядоченного множества |
| in | Проверяет наличие элемента в неупорядоченном множестве |
| issubset | Является ли неупорядоченное множество подмножеством второго неупорядоченного множества |

### 1.4.3 Аналог 3 – Неупорядоченное множество из java8

Особенности реализации неупорядоченного множества в java8:

1. Возможность настраивомого хеширования – имеется возможность указать начальную ёмкость – количество сегментов в хеш-таблице при создании экземпляра класса неупорядоченного множества и коэффициент загрузки – мера того, насколько заполненной должна стать хеш-таблица прежде, чем её ёмкость будет увеличена.
2. Для корректной работы с пользовательским типом данных необходимо переопределить методы “hash” – метод, возвращающий значение хеш-кода в зависимости от объекта хранимого типа и “equals” – метод, возвращающий истину, если 2 элемента хранимого типа равны и ложь в ином случае.

Полный список методов представлен на рисунке 1.

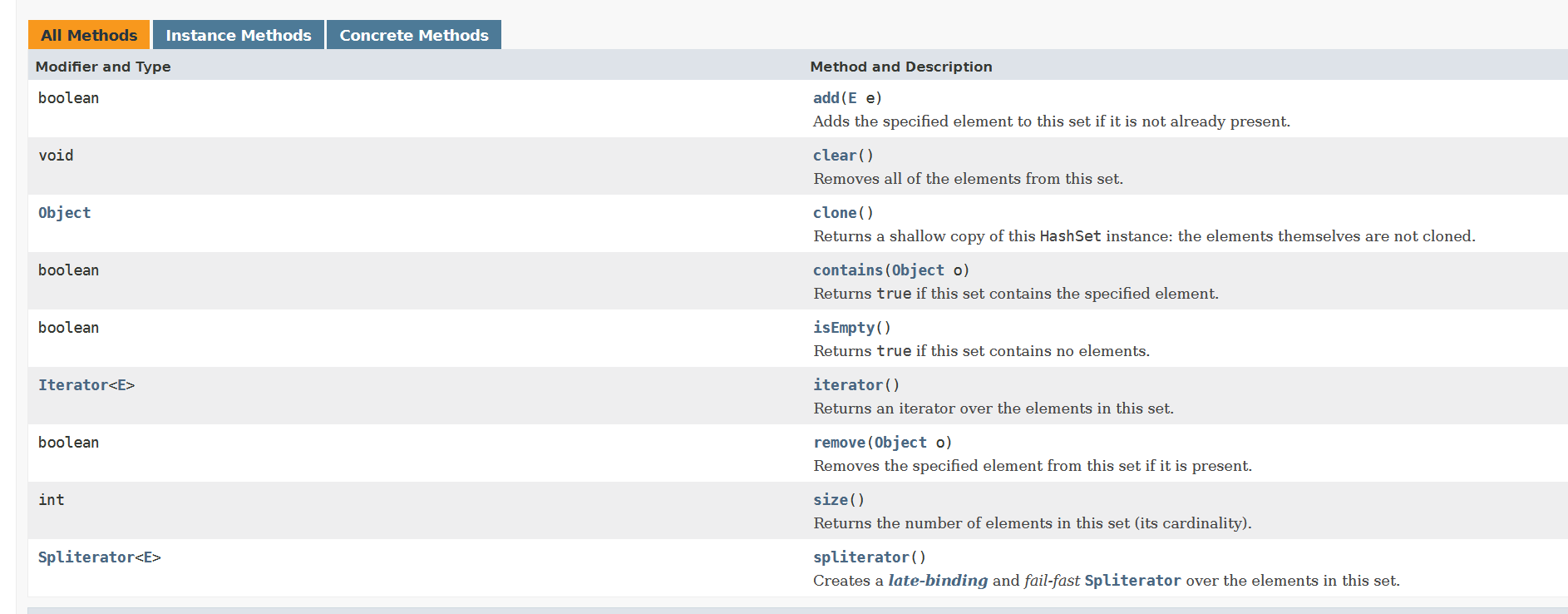


Рисунок 1 – Полный список методов HashSet ЯП java.

## 1.3 Обобщение частной задачи

Написать класс неупорядоченное множество.

После рассмотрения аналогов были выделены следующие методы, которые необходимо будет реализовать:

* Метод для добавления элемента в неупорядоченное множество.
* Метод для удаления элемента из неупорядоченного множества.
* Метод для вывода всех элементов из неупорядоченного множества.
* Метод для копирования одного неупорядоченного множества в другое.
* Метод для проверки наличия элемента в неупорядоченном множестве.
* Метод для получения длины неупорядоченного множества.
* Метод для сравнения двух неупорядоченных множеств.
* Метод для конструирования множества, которое будет включать общие элементы для двух неупорядоченных множеств.
* Метод для конструирования множества, которое будет включать элементы из одного множества и исключать общие элементы с вторым множеством.
* Метод для конструирования множества, которое будет включать элементы из двух множеств.
* Метод для отчистки множества.

# 2. Проектирование

## 2.1 Формальное описание задачи

Написать класс множество на любом языке программирования и написать тестовый модуль для проверки работоспособности класса.

Также имеется возможность выбрать какое множество реализовать – упорядоченное на двоичных деревьях упопоиска или неупорядоченное на хеш-таблицах. Была выбрана реализация неупорядоченного множества на хеш-таблице.

## 2.2 Основные алгоритмы

### 2.2.1 Вставка элемента

Вставка в множество будет происходить в 2 этапа:

* Проверка наличия вставляемого элемента в множестве.
* Вставка элемента по индексу в односвязный список.

Вставка реализуется по следующему алгоритму:

* Вызываем функцию для нахождения hash-кода.
* Создаётся временная переменная для поиска элемента по значению в односвязном списке, соответствующим данному хеш-коду.
* В случае, если элемент уже существует - функция вставки останавливается.
* Если элемент не найден, происходит проверка на то, существует ли хотя бы один элемент с таким же хеш-кодом, если нет – создаётся первый узел односвязного списка.
* Если хоть один элемент найден, значение вставляется в конец односвязного списка.
* В случае успешной вставки значение длины увеличивается на 1.

### 2.2.2 Удаление узла

* Вызывается функция для нахождения hash-кода.
* Создаётся временная переменная для поиска элемента по значению в односвязном списке, соответствующим данному хеш-коду и переменную, хранящую указатель на предыдущий элемент.
* Если элемент найден:
  + Если элемент не единственный с таким значением хеш-кода, то полю предыдущего узла, которое отвечает за указание следующего узла присваивается значение поля текущего узла, отвечающего за указание следующего узла.
  + Если элемент единственный, односвязный список, соответствующий данному хеш-коду, обнуляется.
  + После данных условий текущий узел удаляется.
* В случае успешного удаления значение длины уменьшается на 1.

### 2.2.3 Вывод множества

Вывод множества происходит с помощью прохода по всей хеш-таблице, и отдельного вывода каждого односвязного списка.

Вывод односвязного списка будет происходить с помощью вспомогательного указателя, который изначально будет равен первому элементу (либо ничему если односвязного списка нет с данным хеш-кодом) и сдвигом этого указателя на поле следующий элемент пока этот указатель существует и выводить каждое отдельное значение из узла.

### 2.2.4 Проверка наличия элемента в множестве

* Вызываем функцию для нахождения hash-кода.
* Выполняется обход по односвязному списку как в выводе множества, за исключением того, что вместо вывода каждого отдельного элемента, элементы проверяются на равенство искомому элементу. Если элемент найден будет возвращена истину, иначе ложь.

### 2.2.5 Копирование множеств

Копирование множеств будет реализовано с помощью конструктора копирования.

* Сначала хеш-таблица множества, в которое происходит копирование обнуляется.
* Далее выполняется обход по всем элементам хеш-таблицы множества, из которого будет проходить вставка и каждый элемент вставляется по отдельности в множество, в которое происходит копирование.

### 2.2.6 Объединение множеств

Объединение, пересечение и вычитание множеств будут реализованы следующим образом – метод класса множество, который вернёт новое множество, при этом, не изменяя исходное множество и множество, передаваемое в качестве аргумента метода.

Объединение множеств в теории множеств — множество, содержащее в себе все элементы исходных множеств.

Объединение множеств будет реализовано следующим образом:

* Будет выполнено присвоение результирующему множеству одно из 2х множеств.
* Выполняется обход по всем элементам второго и значение вставляется в результирующие множество, т.к. вставка в множество будет исключать повторения повторов в результирующем множестве не будет.

### 2.2.7 Пересечение множеств

Пересечение множеств в теории множеств — это множество, которому принадлежат те и только те элементы, которые одновременно принадлежат всем данным множествам.

Пересечение множеств будет реализовано следующим образом: выполняется обход по любому множеству и в случае, если элемент находится в другом множестве, вызывается функция вставки для результирующего множества.

### 2.2.8 Разность множеств

Разность двух множеств — теоретико-множественная операция, результатом которой является множество, в которое входят все элементы первого множества, не входящие во второе множество.

Разность множеств будет реализована следующим образом: выполняется обход по множеству, из которого происходит вычитание и в случае, если элемента нет в множестве, из которого происходит вычитание, элемент добавляется в результирующие множество.

### 2.2.9 Сравнение множеств

Сравнение множеств происходит по следующему алгоритму:

* Если длина не равна вернётся ложь.
* Происходит проход по одному из множеств и, если хотя бы один элемент не содержится во втором множестве возвращается ложь.
* Иначе возвращается истина.

### 2.2.10 Отчистка множества

Отчистка множества происходит по следующему алгоритму: для каждого односвязного списка вызывается удаление если первый узел существует (благодаря алгоритмам выше 2 узел сдвигается на место удалённого первого).

### 2.2.11 Проверка множества на пустоту

Если длина множества равна 0, то множество пустое, иначе не пустое.

# 3. Кодирование

## 3.1 Представление данных в памяти при их обработке

Данные будут хранится в хеш-таблице, представленной в виде массива. Обращение будет происходить по хешу элемента. Таким образом, в случае, когда у разных элементов одинаковое значение хеша возникает коллизия. Есть несколько разных способов разрешения коллизий, например:

* Разрешение коллизий с помощью цепочек – каждая ячейка i массива H содержит указатель на начало списка всех элементов, хеш-код которых равен i, либо на их отсутствие.
* Линейное разрешение коллизий - все элементы хранятся непосредственно в хеш-таблице, без использования связных списков. В отличие от хеширования с цепочками, при использовании этого метода может возникнуть ситуация, когда хеш-таблица окажется полностью заполненной, следовательно, будет невозможно добавлять в неё новые элементы. Так что при возникновении такой ситуации решением может быть динамическое увеличение размера хеш-таблицы, с одновременной её перестройкой. [8]

Для разрешения коллизий был выбран метод разрешения коллизий с помощью цепочек.

Для реализации разрешения коллизий методом цепочек был создан вспомогательный класс “узел”, у которого есть 3 поля – указатель на множество, в котором используется узел, значение, хранимое в хеш-таблице и указатель на следующий узел. Определение данного класса можно увидеть в листинге 1.

1. template <typename T>
2. class Node {
3. private:
4. Univercity\_set<T> \*owner;
5. public:
6. T value;
7. Node<T> \*next;
8. Node(T value, Univercity\_set<T> \* owner) { this->value = value; next = NULL; this->owner = owner; };
9. ~Node();
10. };

Листинг 1 – определение класса узел

У основного класса множество 2 поля – защищённый массив указателей, размером, равным размеру максимального хеша минус 1, определение методов, которые необходимо будет реализовать, и поле длина, которое будет хранить текущий размер множества. Определение класса представлено в листинге 2.

1. template <typename T>
2. class Univercity\_set {
3. friend class Node<T>;
4. private:
5. Node<T>\* hash\_table[MAX\_HASH];
6. void out\_value(T value);
7. protected:
8. static std::size\_t hash(T value);
9. Node<T>\* find\_first\_hash(std::size\_t hash);
11. public:
12. int len;
13. void add(T value);
14. void del(T value);
15. bool empty() { return len == 0; };
16. void out();
17. void clear();
18. bool in (T value); // is element in set
20. Univercity\_set union\_set(Univercity\_set another);
21. Univercity\_set intersection\_set(Univercity\_set another);
22. Univercity\_set difference\_set(Univercity\_set another);
24. bool operator==(Univercity\_set &value);
25. Univercity\_set();
26. Univercity\_set(Univercity\_set & cpy); // Univercity\_set dest(src);
27. ~Univercity\_set();
28. };

Листинг 2 – определение класса множество

Описание методов класса:

* Приватный метод out\_value на 16 строке будет отвечать за вывод одного значения.
* Защищённый метод hash на строке 18 будет отвечать за получение хеша в зависимости от элемента.
* Защищённый метод find\_first\_hash на строке 19 будет отвечать за получение первого узла в хеш-таблице по хешу.
* Публичный метод add на строке 23 будет отвечать за добавление элемента в множество.
* Публичный метод del на строке 24 будет отвечать за удаление элемента из множества.
* Публичный метод empty на строке 25 сигнализирует о том, что множество пустое.
* Публичный метод out на строке 26 будет отвечать за вывод множества.
* Публичный метод clear на строке 27 будет отвечать за очистку множества.
* Публичный метод in на строке 28 будет отвечать за проверку наличия элемента в множестве.
* Публичные методы на строках 30-32 будут отвечать за создание нового множества по определённому правилу из 2х множеств.

## 3.2 Структура файлов

Файлы будут хранится в репозитории на [гитхабе](https://github.com/ybivashk4/Coursework_4_sem_set_alghorithms). Сборка будет происходить при помощи Makefile - используются, чтобы помочь решить, какие части большой программы необходимо перекомпилировать. Makefile был назван сборочным файлом. В подавляющем большинстве случаев компилируются файлы C или C++. Множество будет реализовано в файлах .h, подключать нужно только 1 файл – set.h.

set.h – файл, в котором подключаются остальные файлы, исходный код этого файла представлен на рисунке 2.



Рисунок 2 – файл, который будет подключатся в тестовом модуле

Реализация методов класса декомпозирована по файлам .h, список файлов по смыслу:

1. Файл set\_definitions.h будет содержать в себе определение нужных для работы классов и их полей.
2. Файл set\_constructors.h будет содержать в себе реализацию конструкторов - специальный блок инструкций, вызываемый при создании объекта.
3. Файл set\_hash.h будет содержать в себе реализацию методов хеша, которые будут использоваться внутри класса, для ограничения размера хеша – размер хеша не может превышать размер хеш таблицы
4. Файл set\_out.h будет содержать в себе реализацию методов, предназначенных для вывода значений множества.
5. Файл set\_operators.h будет содержать в себе реализацию оператора сравнения 2х множеств.
6. Файл set\_operations.h будет содержать в себе реализацию методов, которые будут выполнять различные операции с множество, а именно:

* Добавление элемента в множество.
* Удаление элемента из множества.
* Очистка множества.
* Проверка входит ли элемент в множество.
* Объединение 2х множеств.
* Пересечение 2х множеств.
* Разность 2х множеств.

Также необходимо создать 2 файла – test.cpp и test.h – файлы для тестирования работоспособности класса.

* Файл test.h будет содержать пользовательский тип данных, чтобы проверить возможность хранить в множестве пользовательский тип данных и макроопределение (функция обработки и замены программного кода) для функции ASSERT и ASSERT\_HINT – функции, которые позволяют указать номер теста и значение, которое должно быть истинным, но если выражение является ложным, это макро-определение выведет в каком файле, строке и функции произошла ошибка, также ASSERT\_HINT может вывести вспомогательное сообщение в случае ошибки. Реализация макроопределения представлена в листинге 3.
* Файл test.cpp будет содержать проверку работоспособности класса множества.

1. #define ASSERT\_HINT(val, hint, ntest) \
2. if (val == false) std::cout << "Error in "\_\_FILE\_\_ << " file, "<<\_\_LINE\_\_ \
3. << " line, " << \_\_FUNCTION\_\_ << " func, "<< hint; \
4. else std::cout << "TEST " << ntest << " success!" << std::endl;
5. #define ASSERT(val, ntest) ASSERT\_HINT(val, NULL, ntest)

Листинг 3 – макроопределение необходимое для тестирования

Общая структура файлов представлена на рисунке 3.

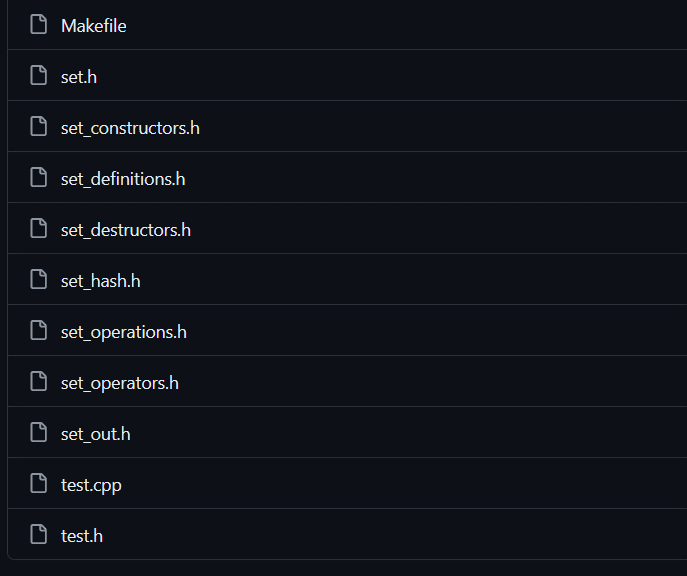


Рисунок 3 - структура файлов программы

## 3.3 Методы класса узел

У класса узел 2 метода – это конструктор и деструктор, их реализация представлена на листингах 4 и 5 соответственно.

1. Node(T value, Univercity\_set<T> \* owner) { this->value = value; next = NULL; this->owner = owner; };

Листинг 4 – конструктор класса узел

1. Node<T>::~Node() {
2. if (next) {
3. Node<T>\* tmp = owner->find\_first\_hash(Univercity\_set<T>::hash(value));
4. if (tmp == this)
5. owner->hash\_table[Univercity\_set<T>::hash(value)] = next;
6. else {
7. while (tmp && tmp->next != this) tmp = tmp->next;
8. if (tmp) tmp->next = this->next;
9. }
10. }
11. }

Листинг 5 – деструктор класса узел

Метод конструктор заполняет значение узла и указывает на множество, в котором создаётся узел.

Деструктор для предыдущего узла изменяет поле, указывающие на следующий элемент, делая его равным следующему узлу для удаляемого элемента.

## 3.4 Методы класса множество

### 3.4.1 Приватные методы класса множество

У класса множество один приватный метод – out\_value, он сделан приватным, т.к. имеет смысл только как вспомогательный метод для метода out.

Функция out\_value реализуется с использованием специализированных шаблонов – для типа int, double и std::string вывод происходит с помощью sttd::cout, а для пользовательского типа данных вывод происходит с помощью метода out, который обязательно должен быть создан в пользовательском типе данных для возможности вывода. Реализация метода представлена на листинге 6.

1. template <typename T>
2. void Univercity\_set<T>::out\_value(T value) {
3. Node<T> \* tmp = hash\_table[Univercity\_set<T>::hash(value)];
4. while (tmp == nullptr && !(tmp->value == value))
5. tmp=tmp->next;
6. if (tmp) tmp->value.out();
7. else std::cout << "(null)";
8. }
9. template <>
10. void Univercity\_set<int>::out\_value(int value) {
11. std::cout << value;
12. }
13. template <>
14. void Univercity\_set<double>::out\_value(double value) {
15. std::cout << value;
16. }
17. template <>
18. void Univercity\_set<std::string>::out\_value(std::string value) {
19. std::cout << value;
20. }

Листинг 6 – Метод out\_value

### 3.4.2 Защищённые методы класса множество

Объявление защищённых методов представлено на листинге 7.

1. static std::size\_t hash(T value);
2. Node<T>\* find\_first\_hash(std::size\_t hash);

Листинг 7 – объявление защищённых методов

Метод hash не зависит от экземпляра класса, поэтому он был сделан статическим. Данный метод возвращает хеш, ограниченный максимальным размером хеша, также с помощью специализированных шаблонов были реализованы свои варианты хеширования для 3х типов данных – int, double и std::string. Для int и double сначала идёт преобразование к беззнаковому целому, после чего выполняется побитовое исключающие ИЛИ с задаваемой маской. Для пользовательского же типа данных вызывается реализованный внутри типа метода для получения хеша и ограничивается. Реализация метода hash представлена на листинге 8.

1. template <typename T>
2. std::size\_t Univercity\_set<T>::hash(T value) {
3. return value.hash() % (MAX\_HASH);
4. }
5. template <>
6. std::size\_t Univercity\_set<int>::hash(int value) {
7. return (std::size\_t)(value ^ HASH\_MASK\_FOR\_INT) % (MAX\_HASH);
8. }
9. template <>
10. std::size\_t Univercity\_set<double>::hash(double value) {
11. return ((unsigned int)value ^ HASH\_MASK\_FOR\_INT) % (MAX\_HASH);
12. }
13. template <>
14. std::size\_t Univercity\_set<std::string>::hash(std::string value) {
15. return ((sizeof(value) + (value[0] + value[value.length() - 1]) / 2) % (MAX\_HASH));
16. }

Листинг 8 – реализация метода hash

Метод find\_first\_hash необходим для деструктора класса Node, данный метод возвращает указатель на первый элемент односвязного списка, соответствующего переданному хешу. Реализация представлена в листинге 9.

1. template <typename T>
2. Node<T>\* Univercity\_set<T>::find\_first\_hash(std::size\_t hash) {
3. return hash\_table[hash];
4. }

Листинг 9 – Реализация метода find\_first\_hash

### 3.4.3 Публичные методы класса множество

Публичные методы соответствуют пункту 2.5 “основные алгоритмы” в проектировании. Также добавляется 2 метода – конструктор и деструктор для множества.

Объявление методов представлено на листинге 10.

1. void add(T value);
2. void del(T value);
3. bool empty() { return len == 0; };
4. void out();
5. void clear();
6. bool in (T value); // is element in set
8. Univercity\_set union\_set(Univercity\_set another);
9. Univercity\_set intersection\_set(Univercity\_set another);
10. Univercity\_set difference\_set(Univercity\_set another);
12. bool operator==(Univercity\_set &value);
13. Univercity\_set();
14. Univercity\_set(Univercity\_set & cpy); // Univercity\_set dest(src);
15. ~Univercity\_set();

Листинг 10 – публичные методы класса множество

Метод add действует по алгоритму из п. 2.2.1, реализация представлена в листинге 11.

1. template <typename T>
2. void Univercity\_set<T>::add(T value) {
3. int index = Univercity\_set<T>::hash(value);
4. Node<T>\* tmp = hash\_table[index];
5. while (tmp) {
6. if (tmp->value == value) {
7. return;
8. }
9. tmp = tmp->next;
10. }
11. tmp = hash\_table[index];
12. if (!hash\_table[index])
13. hash\_table[index] = new Node<T>(value, this);
14. else {
15. while (tmp->next) {
16. tmp = tmp->next;
17. }
18. tmp->next = new Node<T>(value, this);
19. }
20. len++;
21. }

Листинг 11 – реализация метода вставки в множество

На строчке 128 получается значение хеш-кода для данного элемента, далее на строчках 129 – 135 идёт поиск элемента с таким же значением, после чего на строчках 136-144 происходит вставка элемента в множество, на строке 145 значение поля длина увеличивается.

Метод del действует по алгоритму из п. 2.2.2, реализация представлена в листинге 12.

1. template <typename T>
2. void Univercity\_set<T>::del(T value) {
3. int index = Univercity\_set<T>::hash(value);
4. if (hash\_table[index]) {
5. Node<T>\* tmp = hash\_table[index];
6. Node<T>\* prev = nullptr;
7. while (tmp && !(tmp->value == value)) {
8. prev = tmp;
9. tmp = tmp->next;
10. }
11. if (tmp) {
12. if (prev) {
13. prev->next = tmp->next;
14. } else {
15. hash\_table[index] = tmp->next;
16. }
17. delete tmp;
18. len--;
19. }
20. }
21. }
22. template <>
23. void Univercity\_set<double>::del(double value) {
24. int index = Univercity\_set<double>::hash(value);
25. if (hash\_table[index]) {
26. Node<double>\* tmp = hash\_table[index];
27. Node<double>\* prev = nullptr;
28. while (tmp && fabs(tmp->value - value) >= 1e-6 ) {
29. prev = tmp;
30. tmp = tmp->next;
31. }
32. if (tmp) {
33. if (prev) {
34. prev->next = tmp->next;
35. } else {
36. hash\_table[index] = tmp->next;
37. }
38. delete tmp;
39. len--;
40. }
41. }
42. }

Листинг 12 – реализация удаления элемента из множества

В связи с тем, что числа с плавающей запятой небезопасно сравнивать с помощью обычно оператора сравнения, была создана реализация метода с специализированным для double шаблоном, где проверка происходит при помощи сравнения с эпсилон, равной 1e-6, происходит сравнение на строке 178. В остальном методы удаления идентичны.

На строке 150 можно видеть получение хеш-кода для удаляемого элемента. На строках 151 и 152 создаются указатель на текущий узел и на предыдущий. Далее алгоритм выполняется на строках 154-165.

Метод out действует по алгоритму из п. 2.2.3, реализация представлена в листинге 13.

1. template <typename T>
2. void Univercity\_set<T>::out() {
3. for (int i = 0; i < MAX\_HASH; i++) {
4. if (hash\_table[i]) {
5. Node<T> \* tmp = hash\_table[i];
6. while (tmp) {
7. out\_value(tmp->value);
8. std::cout << " ";
9. tmp = tmp->next;
10. }
11. }
12. }
13. }
14. template <>
15. void Univercity\_set<int>::out() {
16. for (int i = 0; i < MAX\_HASH; i++) {
17. Node<int> \* tmp = hash\_table[i];
18. while (tmp) {
19. out\_value(tmp->value);
20. std::cout << " ";
21. tmp = tmp->next;
22. }
23. }
24. std::cout << std::endl;
25. }
26. template <>
27. void Univercity\_set<double>::out() {
28. for (int i = 0; i < MAX\_HASH; i++) {
29. if (hash\_table[i]) {
30. Node<double> \* tmp = hash\_table[i];
31. while (tmp) {
32. out\_value(tmp->value);
33. std::cout << " ";
34. tmp = tmp->next;
35. }
36. }
37. }
38. std::cout << std::endl;
39. }
40. template <>
41. void Univercity\_set<std::string>::out() {
42. for (int i = 0; i < MAX\_HASH; i++) {
43. if (hash\_table[i]) {
44. Node<std::string> \* tmp = hash\_table[i];
45. while (tmp) {
46. out\_value(tmp->value);
47. std::cout << " ";
48. tmp = tmp->next;
49. }
50. }
51. }
52. std::cout << std::endl;
53. }

Листинг 13 – реализация метода out

Для пользовательских типов данных не выводиться конец строки после вывода, чтобы не дублировать вывод конца строки. Поэтому для типов данных, которые не обладают пользовательским выводом создана отдельная реализация.

Вывод происходит при помощи описанного ранее метода out\_value.

Метод in действует по алгоритму из п. 2.2.4, реализация представлена в листинге 14.

1. template <typename T>
2. bool Univercity\_set<T>::in(T value) {
3. std::size\_t index = Univercity\_set<T>::hash(value);
4. if (!hash\_table[index]) return false;
5. Node<T> \* tmp = hash\_table[index];
6. while (tmp) {
7. if (tmp->value == value) return true;
8. tmp = tmp->next;
9. }
10. return false;
11. }

Листинг 14 – реализация метода in

Метод Univercity\_set(Univercity\_set & cpy) – конструктор копирования. Данный метод действует по алгоритму из п. 2.2.5, реализация представлена в листинге 15.

1. template <typename T>
2. Univercity\_set<T>::Univercity\_set(Univercity\_set & cpy) {
3. len = 0;
4. for (int i=0;i<MAX\_HASH;i++) hash\_table[i] = nullptr;
5. for (int i = 0; i < MAX\_HASH; i++) {
6. Node<T> \* tmp = cpy.hash\_table[i];
8. while (tmp) {
9. add(tmp->value);
10. tmp = tmp->next;
11. }
12. }
13. }

Листинг 15 – реализация конструктора копирования для класса множество

Метод union\_set действует по алгоритму из п. 2.2.6, реализация представлена в листинге 16.

1. template <typename T>
2. Univercity\_set<T> Univercity\_set<T>::union\_set(Univercity\_set<T> another) {
3. Univercity\_set<T> res = another;
4. for (int i = 0; i < MAX\_HASH; i++) {
5. Node<T> \* tmp = hash\_table[i];
6. while (tmp) {
7. res.add(tmp->value);
8. tmp = tmp->next;
9. }
10. }
11. return res;
12. }

Листинг 16 – реализация метода объединения множеств

Метод intersection\_set действует по алгоритму из п. 2.2.7, реализация представлена в листинге 17.

1. template <typename T>
2. Univercity\_set<T> Univercity\_set<T>::intersection\_set(Univercity\_set<T> another) {
3. Univercity\_set<T> res;
5. for (int i = 0; i < MAX\_HASH; i++) {
6. Node<T> \* tmp = hash\_table[i];
7. while (tmp) {
8. if (another.in(tmp->value)) res.add(tmp->value);
9. tmp = tmp->next;
10. }
11. }
12. return res;
13. }

Листинг 17 – реализация метода пересечения множеств

Метод difference\_set действует по алгоритму из п. 2.2.8, реализация представлена в листинге 18.

1. template <typename T>
2. Univercity\_set<T> Univercity\_set<T>::difference\_set(Univercity\_set<T> another) {
3. Univercity\_set<T> res;
4. for (int i = 0; i < MAX\_HASH; i++) {
5. Node<T> \* tmp = hash\_table[i];
6. while (tmp) {
7. if (!another.in(tmp->value)) res.add(tmp->value);
8. tmp = tmp->next;
9. }
10. }
11. return res;
12. }

Листинг 18 – реализация метода объединения множеств

## 3.5 Разработка кода сборочного файла

* Каждый .h файл начинается на #ifndef, #define SET\_... и заканчивается на #endif, это сделано, чтобы была возможность подключать одни и те же .h файлы друг в друга, чтобы при этом если файл включился он был включён 1 раз. Например, файл set\_definitions.h включается во все файлы, т.к. каждый файл работает с каким-либо классом.
* Сборка проекта происходит с помощью сборочного файла. Код сборочного файла можно увидеть в листинге 19. Цель lcov\_report у сборочного файла будет использоваться для получения html-документа, который покажет, насколько тесты хорошо покрывают классы и методы.
* Полный код программы можно найти в приложениях Б.1-Б.9.

1. CC := g++
2. SOURCES := $(wildcard \*.cpp)
3. BUILDDIR := .
4. SOURCEDIR := .
5. OBJECTS := $(SOURCES:.cpp=.o)
6. TARGET := test
7. CFLAGS :=
8. RFLAGS := --coverage
9. all : $(TARGET)
10. $(TARGET) : $(OBJECTS)
11. # $@ - имя цели, %< - файлы после :
12. $(BUILDDIR)/%.o: $(SOURCEDIR)/%.cpp
13. $(CC) -c $(CFLAGS) $< -o $@ $(RFLAGS)
14. clean :
15. rm -rf \*.o \*.gcno \*.gcda \*.out \*.info $(TARGET)
16. rm -rf cvg
17. rm -rf results
18. lcov\_report : clean
19. g++ test.cpp --coverage
20. ./a.out
21. mkdir cvg
22. cd cvg
23. lcov -c -d ../ -o my.info
24. genhtml -o results/ my.info
25. rebuild : clean all

Листинг 19 – содержание сборочного файла

# 4. Тестирование

## 4.1 Поведенческая модель программы

Модуль для проверки работоспособности класса был назван тестовым модулем.

Тестовый модуль подключает файл, в котором лежит класс и файлы необходимые для тестирования.

Далее идёт определение функций, которые будут задействованы для тестирования в процессе работы программы.

В основной функции тестирования сначала будет произведено базовое тестирование для разных типов. Выполнено это в связи с реализацией, которая допускает использование только 3х встроенных в ЯП типов – целочисленное знаковое число (int), знаковое число с плавающей точкой (double) и тип позволяющий удобно работать с строками (std::string). Помимо приведённых выше типов можно использовать любой свой тип, у которого будут выполнятся следующие условия:

1. Получить хеш с помощью соответствующего метода.
2. Будет возможность сравнивать экземпляры этого типа на равенство.
3. Будет возможность вывести элементы этого типа.

Пример подходящего типа представлен на листинге 20. В данной структуре.

* Условие 1 выполняется на строчке 344.
* Условие 2 на строчках 348-350.
* Условие 3 на строчках 345-347.

1. struct Coordinates {
2. int x, y, z;
3. Coordinates() : x(0), y(0), z(0){};
4. Coordinates(int x\_, int y\_, int z\_) : x(x\_), y(y\_), z(z\_){};
5. std::size\_t hash() {return (x \* y \* z);};
6. void out() {
7. std::cout << x << " " << y << " " << z << std::endl;
8. }
9. bool operator == (const Coordinates & cmp) {
10. return (x == cmp.x && y == cmp.y && z == cmp.z);
11. }
12. };

Листинг 20 – пример типа данных, который можно хранить в множестве

## 4.2 Проверка работоспособности класса

Для того, чтобы проверить работоспособность класса, было использовано макроопределение, приведённое в листинге 3 на 16 стр. и отдельно проверка вывода и базовых операций для разных типов без использования макроопределения. Полный код модуля тестирования представлен в приложении Б.10. Результат работы модуля представлен на рисунке 4.

Вначале программы происходит создание экземпляров класса множество с разными типами данных – int, double, std::string, Coordinates. Тип данных Coordinates пользовательский и подходит для хранения в множестве. Строки 432-435.

Далее первый тест – попытка вывода пустого множества, программа выводит 3 пустые строки. Строка 437.

Далее на строке 440 происходит заполнение множеств некоторыми различными значениями.

Далее на строке 442 происходит вывод непустых множеств.

Далее на строке 445 происходит удаление одного существующего элемента из каждого множества и вывод на строке 446.

Итог первого теста представлен в таблице 4.

Удаляемые значения:

* int: 1
* double : 2.1
* std::string : asd
* Coordinates : (2e9 3456464 475 )

Таблица 4 – результат первого теста

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Состояние/тип | int | double | std::string | Coordinates |
| До удаления | 3 2 1 | 3.1 2.1 1.1 | asd dsa zxc | 1 2 3  2 3 4  2e9 3456464 475  9 7 5 |
| После удаления | 3 2 | 3.1 1.1 | dsa zxc | 1 2 3  2 3 4  9 7 5 |

Далее в программе выполняется 13 тестов, которые проверяют больше аспектов класса множество на примере множества целых чисел.

* Тест на строках 452-457 необходим для проверки, что в множество не получится добавить одинаковый элемент. Множество после теста – “3 2”.
* На строках 460-465 тест проверяет возможность добавлять в множество элемент с одинаковым хеш-кодом. Значение с таким же хеш-кодом были найдены с помощью скрипта на ЯП python3 код программы представлен в листинге 21. Данный скрипт ищет 5 чисел после числа 2, которые дадут такой же хеш-код, как и 2-ка. Выводом программы были следующие числа – 1002, 1050, 2098, 3146, 4194. Множество после теста – “3 2 1002 1050 2098 3146 4194”.
* На строках 467-471 проверяется возможность удалять элементы с одинаковым хеш-кодом. Множество после теста – “3 2 1050 3146”.
* На строках 474-477 проверяется поведение множества в случае, когда происходит попытка удалить несуществующий элемент. Множество после теста – “3 2 1050 3146”.
* На строках 480-484 проверяется метод is\_empty и удаление последнего элемента из множества. Множество после теста – “”.
* На строках 486-491 проверяется метод clear. Множество после теста – “”.
* На строках 494-501 проверяется конструктор копирования. Множество после теста – “3 2 1002 3146”, Тестовое множество после теста – “3 2 1002 3146”.
* На строке 504 проверяется метод, проверяющий вхождение элемента в множество. Множество после теста – “3 2 1002 3146”, Тестовое множество после теста – “3 2 1002 3146”.
* На строке 505 проверяется метод сравнения множеств. Множество после теста – “3 2 1002 3146”, Тестовое множество после теста – “3 2 1002 3146”.
* На строках 506-513 проверяется пересечение множеств. Множество после теста – “3 2 1002 3146 1 1001 9”, Тестовое множество после теста – “3 2 1002 3146 666”, Результирующие множество – “3 2 1002 3146”.
* На строках 516-518 проверяется разность множеств. Множество после теста – “3 2 1002 3146 1 1001 9”, Тестовое множество после теста – “3 2 1002 3146 666”, Результирующие множество – “3 2 1002 3146”.
* На строках 520-524 проверяется объединение множеств. Множество после теста – “3 2 1002 3146 1 1001 9”, Тестовое множество после теста – “3 2 1002 3146 666”, Результирующие множество – “9 3 2 1002 1 1001 3146 666”.

1. cnt = 0
2. for i in range(3, 1000000):
3. if ((i ^ 1023)% 1000 == (2^1023) % 1000):
4. print(i)
5. cnt+=1
6. if (cnt == 5):
7. break

Листинг 21 – поиск чисел с одинаковым хеш-кодом

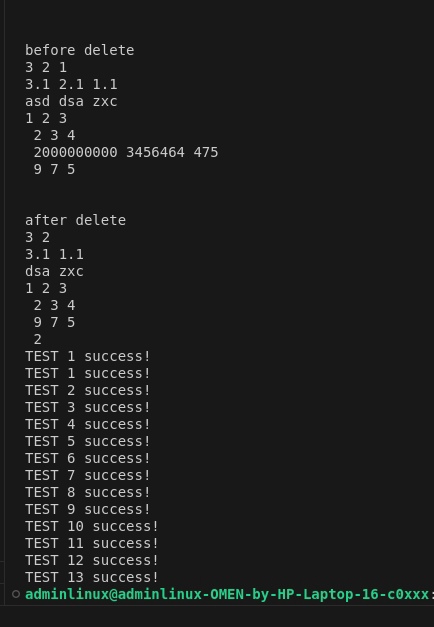


Рисунок 4 – вывод тестового модуля

Для проверки общего качества тестирования была использована утилита LCOV — это расширение GCOV, инструмента GNU, который предоставляет информацию о том, какие части программы фактически выполняются (т. е. «покрываются») во время выполнения конкретного тестового примера. [9]

С помощью данной утилиты был создан файл с расширением html, показывающий покрытие тестами. Результат работы утилиты LCOV представлен на рисунке 5. Реализована данная утилита в сборочном файле в листинге 19, строчках 331 – 337. Чтобы получить результат работы данной утилиты в командной строке необходимо прописать **make lcov\_report**. Также необходимы установленные Make и lcov. На рисунке 5 столбец line coverage означает количество строк, которые отработали в результате работы тестового модуля. Functions означает количество вызванных функций.

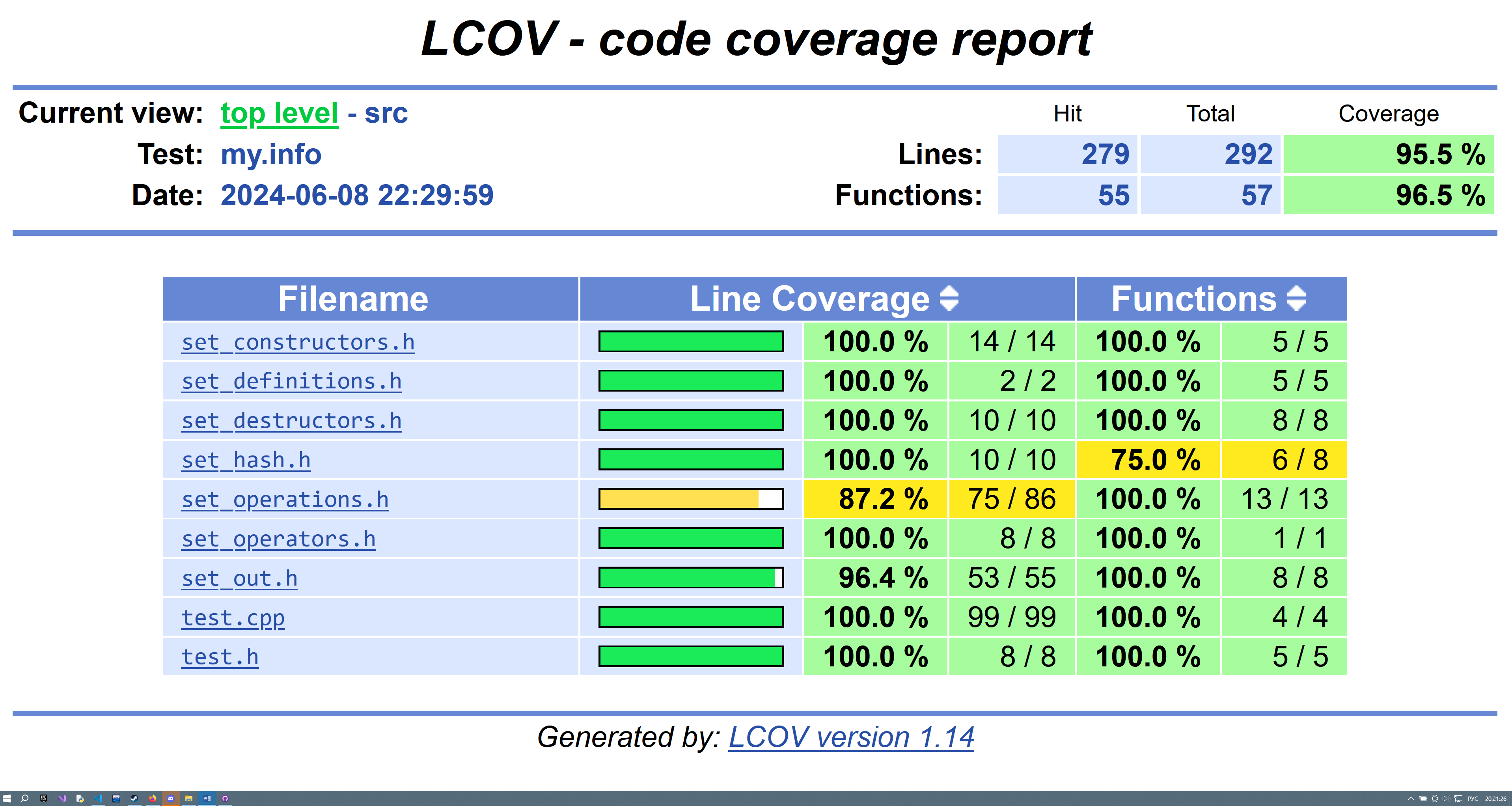


Рисунок 5 – результат работы утилиты LCOV

# 5. Исследование операции вставки во множество

Для проверки быстродействия класса был использован код, реализованный в ходе выполнения лабораторной работы №1 по дисциплине структуры и алгоритмы обработки данных. Использовался класс double\_array – класс, который хранит в себе последовательность определённого размера и позволяет генерировать различные последовательности. Для проверки использовались следующие генерации:

* Генерация упорядоченной по возрастанию последовательности - метод gen\_up(min\_value, max\_value, step), данный метод начиная с min\_value генерирует последовательность, увеличивая каждый следующий элемент на step. max\_value – максимально возможное число для генерации.
* Генерация случайной последовательности - метод gen\_rand(min\_value, max\_value), данный метод генерирует последовательность с случайными элементами в диапазоне от min\_value до max\_value.
* Генерация квази-упорядоченной последовательности (частично упорядоченной последовательности) – метод gen\_kvazi(min\_value, max\_value) – создаёт квази-упорядоченную последовательность в диапазоне от min\_value до max\_value.

Полный код данной лабораторной работы представлен в 2х файлах -lab1.cpp и lab1.h в приложениях Б.11 и Б.12.

Для получения результатов тестирования программа запускалась 3 раза, при вставке из квази-упорядоченной и случайно-упорядоченной последовательности программа вывела одну таблицу данных, при вставке из упорядоченной программа вывела 2 таблицы – результат при генерации с шагом 10 и шагом 0.93.

В ходе тестирование выяснилось, что размер хеш-таблицы, равный 1000 сильно замедляет работу множества по сравнению с размером хеш-таблицы равным 100000 вплоть до 2х порядков. Происходит данная ситуация из-за того, что метод устранения коллизий цепочками в случае, когда количество элементов, хранимых в множестве сильно выше значения максимального хеша. В итоге тестирование проводилось с размером хеш-таблицы равным 100000 (вместо строки 30 приложения Б.2 была написана строка *#define MAX\_HASH 100000*).

Модуль, с помощью которого замерялось время представлен в приложении Б.13. В данном коде закомментированные вызовы, чтобы запустить программу нужно раскомментировать нужный сценарий.

Результаты запусков модуля представлены в таблицах 5-8 и рисунках 6-9.

Таблица 5 – время работы операции вставки из случайно сгенерированной последовательности

|  |  |  |
| --- | --- | --- |
|  | Реализация множества в курсовой работе | Реализация множества в стандартной библиотеке шаблонов c++ |
| Количество элементов | Время, сек | |
| 1е7 | 0,067813 | 0,542459 |
| 3е7 | 0,198058 | 1,63728 |
| 5е7 | 0,323744 | 2,69983 |
| 7е7 | 0,449824 | 3,76535 |
| 9е7 | 0,580222 | 4,84672 |
| 1е8 | 0,64454 | 5,38068 |

Таблица 6 – время работы операции вставки из упорядоченной последовательности с шагом меньше 1

|  |  |  |
| --- | --- | --- |
|  | Реализация множества в курсовой работе | Реализация множества в стандартной библиотеке шаблонов c++ |
| Количество элементов | Время, сек | |
| 1е6 | 0,005222 | 0,060205 |
| 3е6 | 0,015617 | 0,177254 |
| 5е6 | 0,025667 | 0,293608 |
| 7е6 | 0,0354 | 0,409693 |
| 9е6 | 0,045604 | 0,532435 |
| 1е7 | 0,051829 | 0,599859 |

Таблица 7 – время работы операции вставки из упорядоченной последовательности с шагом равным 10

|  |  |  |
| --- | --- | --- |
|  | Реализация множества в курсовой работе | Реализация множества в стандартной библиотеке шаблонов c++ |
| Количество элементов | Время, сек | |
| 1е6 | 0,08055 | 0,471579 |
| 3е6 | 1,60736 | 1,98903 |
| 5е6 | 4,16761 | 2,75529 |
| 7е6 | 7,27029 | 4,54556 |
| 9е6 | 12,6797 | 5,47957 |
| 1е7 | 15,4298 | 5,85249 |

Таблица 8 – время работы операции вставки из квази-упорядоченной последовательности

|  |  |  |
| --- | --- | --- |
|  | Реализация множества в курсовой работе | Реализация множества в стандартной библиотеке шаблонов c++ |
| Количество элементов | Время, сек | |
| 1е6 | 0,04612 | 0,479221 |
| 3е6 | 0,202869 | 1,94112 |
| 5е6 | 0,508989 | 2,76976 |
| 7е6 | 1,15294 | 4,62798 |
| 9е6 | 2,36048 | 5,44536 |
| 1е7 | 2,52993 | 5,89972 |

Рисунок 6 - график зависимости времени от количества элементов для вставки элементов из случайной последовательности.

Рисунок 7 - график зависимости времени от количества элементов для вставки элементов из упорядоченной последовательности с шагом меньше 1.

Рисунок 8 - график зависимости времени от количества элементов для вставки элементов из упорядоченной последовательности с шагом 10

Рисунок 9 - график зависимости времени от количества элементов для вставки элементов из квази-упорядоченной последовательности

Итоги:

В первую очередь стоит сказать, что плюс реализации неупорядоченного множества в данной работе заключается в том, что можно жертвовать памятью, чтобы увеличить скорость работы программы и наоборот. Данное утверждение следует из того, что при увеличении размера хеш-таблицы до 100000 класс может составить конкуренцию реализации класса неупорядоченного множества из STL. Но рисунок 8 показывает, что данная реализация не идеальна и в некоторых ситуациях может проиграть по скорости неупорядоченному множеству из STL.

Превосходство в скорости на порядок данной реализации над реализацией STL при случайной генерации происходит из-за того, что в STL в качестве хеша используются функция, у которой вероятность того, что разные значения будут иметь одинаковый хеш равна 1/ 4294967295.[10] В то время как в реализации класса неупорядоченное множество в данной работе вероятность не контролируется и, следовательно, в большинстве случаев будет много выше, чем в реализации STL.

Класс работает достаточно стабильно, это показывает покрытие LCOV на рисунке 5.

# Заключение

Реализован класс неупорядоченное множество на языке программирования “С++”. Также реализовано много тестов для доказательства полезности и надёжности класса.

Программу можно изменить – класс станет занимать больше памяти, но время работы будет медленнее расти при увеличении количества элементов.

Поставленная цель в задании выполнена.

# Список использованных источников

1. GeeksforGeeks, Introduction to Tree – Data Structure and Algorithm Tutorials (Введение в множество – Учебные пособия по алгоритмам и структурам) - https://www.geeksforgeeks.org/introduction-to-set-data-structure-and-algorithm-tutorials/

2. GeeksforGeeks,

Applications, Advantages and Disadvantages of Set Применения, достоинства и недостатки структуры данных множество) –

<https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-set/>

3. GeeksforGeeks, Сравнение упорядоченного и неупорядоченного множества в c++, STL

<https://www.geeksforgeeks.org/set-vs-unordered_set-c-stl/>

4. cppreference, Справочник по C, C++, неупорядоченное множество –

https://en.cppreference.com/w/cpp/container/unordered\_set

5. python doc, встроенные типы

https://docs.python.org/3/library/stdtypes.html#set

6. Java Platform Standard Edition 8 Documentation, класс HashSet –

https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html

7. *Вирт Н.* Алгоритмы и структуры данных. — М.: Мир, 1989

8. ИТМО конспекты, Разрешение коллизий в хеш таблице. <https://neerc.ifmo.ru/wiki/index.php?title=Разрешение_коллизий>

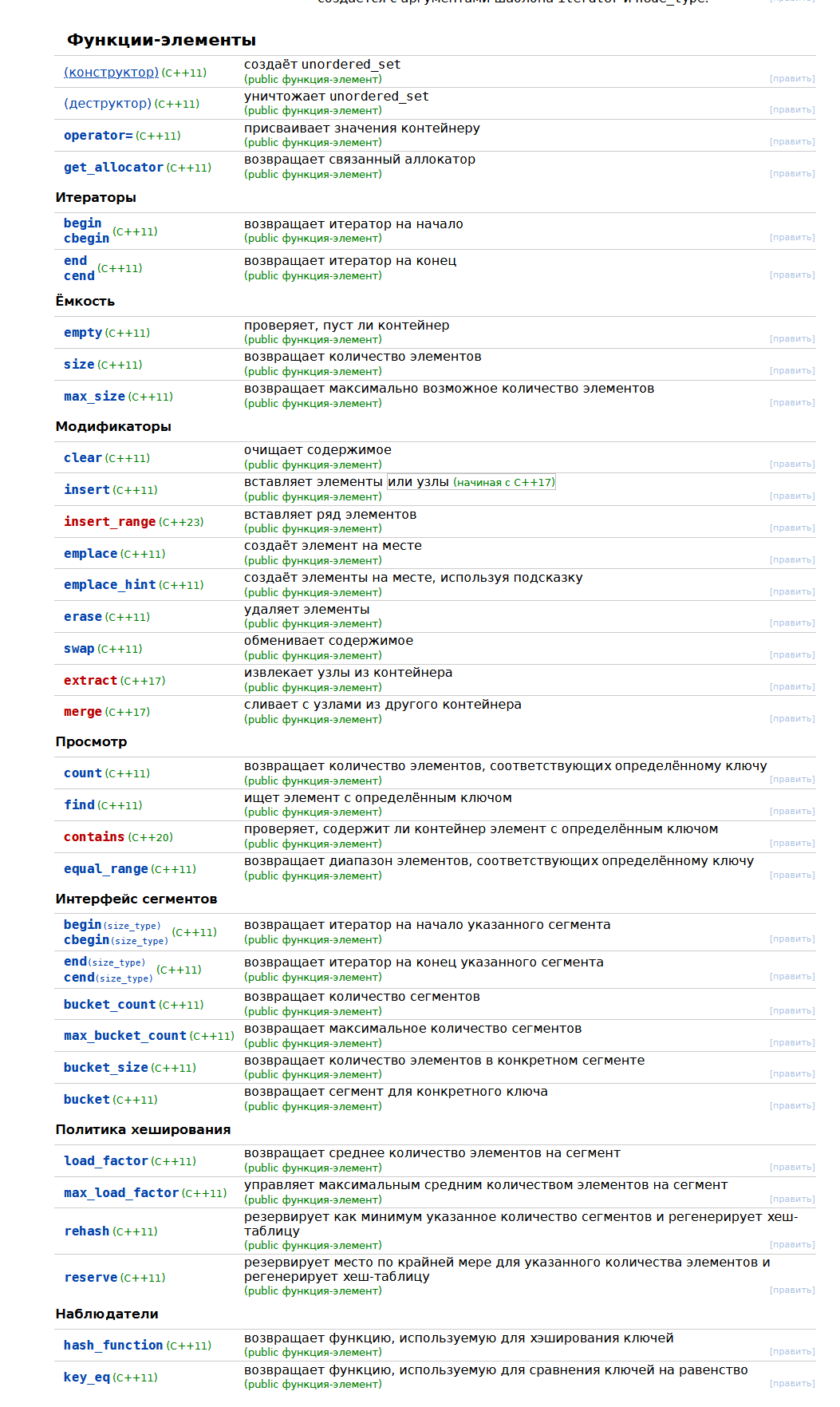
9. Документация для утилиты LCOV - https://lcov.readthedocs.io/en/latest/

10. cppreference.com- Документация для ЯП c++, функция hash, <https://en.cppreference.com/w/cpp/named_req/Hash>

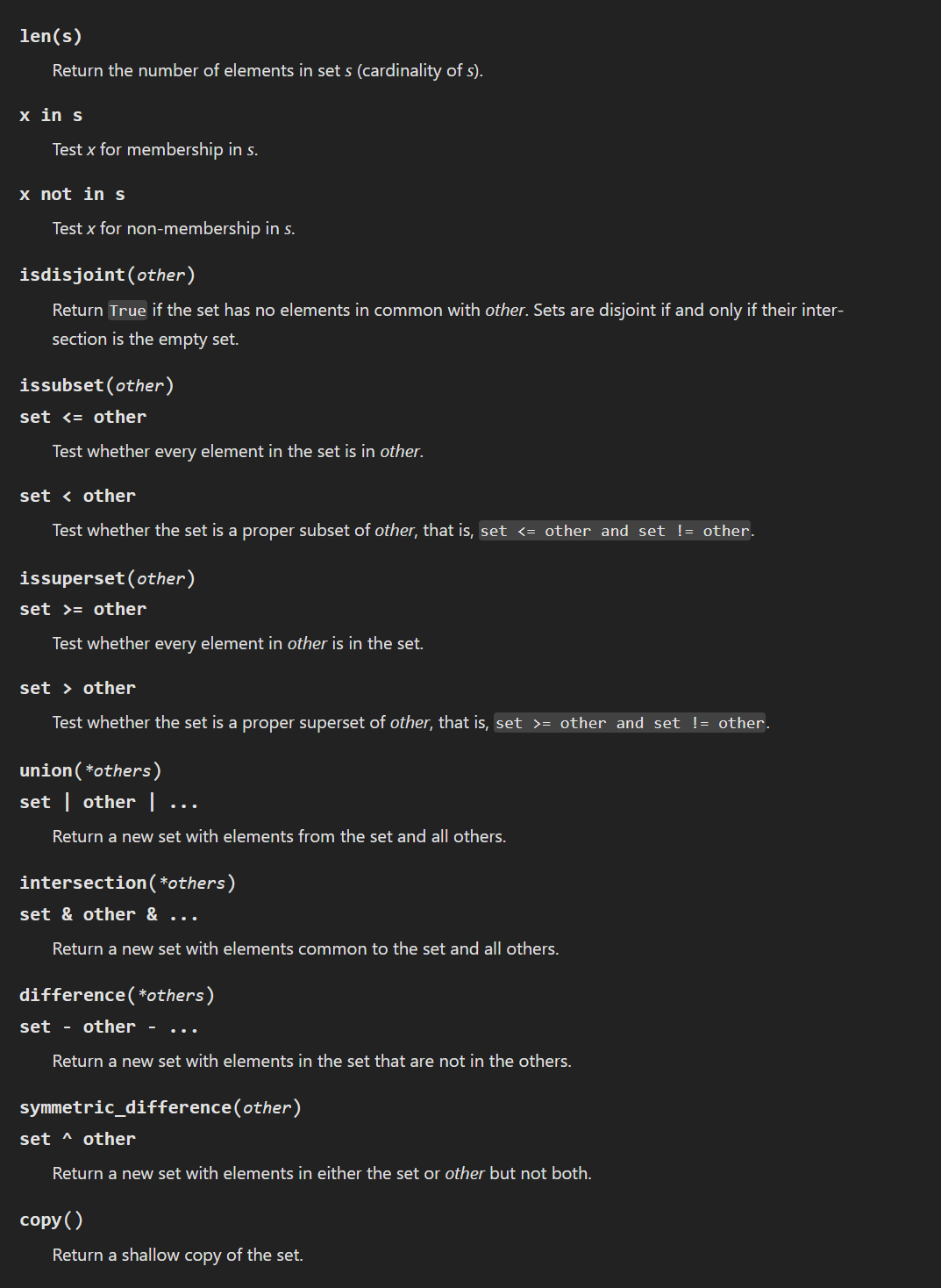
# Приложения

## Приложения А. Рисунки

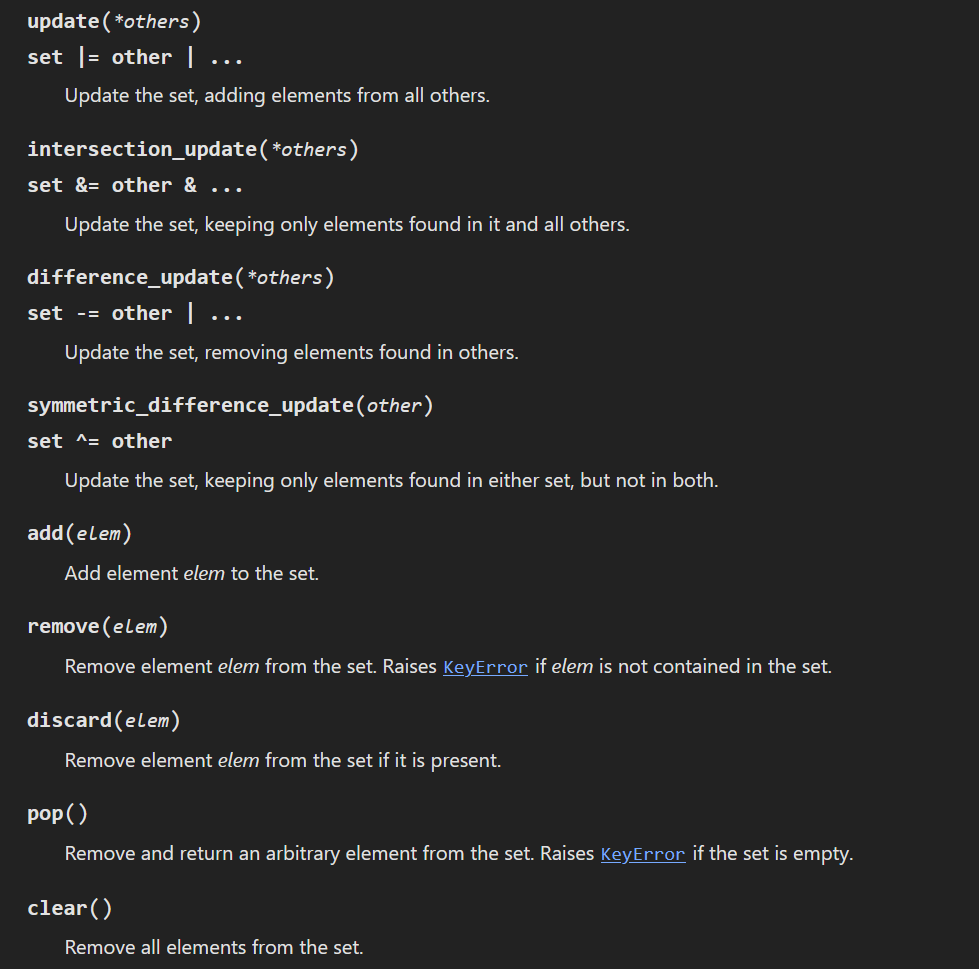
Приложение А.1 – методы класса неупорядоченное множество в С++



Приложение А.2.1 – первая половина методов в неупорядоченном множестве в python3



Приложение А.2.2 – вторая половина методов в неупорядоченном множестве в python3



## Приложения Б. Листинги

Приложение Б.1 – файл set\_constructors.h – реализация всех конструкторов, используемых в множестве.

1. #ifndef SET\_CONSTRUCTORS
2. #define SET\_CONSTRUCTORS
3. #include "set\_definitions.h"
4. template <typename T>
5. Univercity\_set<T>::Univercity\_set() {
6. for (int i = 0; i < MAX\_HASH; i++) {
7. hash\_table[i] = nullptr;
8. }
9. len = 0;
10. }
11. template <typename T>
12. Univercity\_set<T>::Univercity\_set(Univercity\_set & cpy) {
13. len = 0;
14. for (int i=0;i<MAX\_HASH;i++) hash\_table[i] = nullptr;
15. for (int i = 0; i < MAX\_HASH; i++) {
16. Node<T> \* tmp = cpy.hash\_table[i];
18. while (tmp) {
19. add(tmp->value);
20. tmp = tmp->next;
21. }
22. }
23. }
24. #endif

Приложение Б.2 – файл set\_definitions.h – файл, содержащий определение всех используемых в множестве классов, методов классов и макроопределений.

1. #ifndef SET\_DEF
2. #define SET\_DEF
3. #define MAX\_HASH 1000
4. #define HASH\_MASK\_FOR\_INT 1023
5. #include <string>
6. #include <cmath>
7. template <typename T> class Univercity\_set;
8. template <typename T> class Node;
9. template <typename T>
10. class Node {
11. private:
12. Univercity\_set<T> \*owner;
13. public:
14. T value;
15. Node<T> \*next;
16. Node(T value, Univercity\_set<T> \* owner) { this->value = value; next = NULL; this->owner = owner; };
17. ~Node();
18. };
19. /\*
20. if you use custom structure or class in Univercity\_set must be
21. 1) method size\_t hash();
22. 2) operator ==
23. 3) method <anything> out();
24. \*/
25. template <typename T>
26. class Univercity\_set {
27. friend class Node<T>;
28. private:
29. Node<T>\* hash\_table[MAX\_HASH];
30. void out\_value(T value);
31. protected:
32. static std::size\_t hash(T value);
33. Node<T>\* find\_first\_hash(std::size\_t hash);
35. public:
36. int len;
37. void add(T value);
38. void del(T value);
39. bool empty() { return len == 0; };
40. void out();
41. void clear();
42. bool in (T value); // is element in set
44. Univercity\_set union\_set(Univercity\_set another);
45. Univercity\_set intersection\_set(Univercity\_set another);
46. Univercity\_set difference\_set(Univercity\_set another);
48. bool operator==(Univercity\_set &value);
49. Univercity\_set();
50. Univercity\_set(Univercity\_set & cpy); // Univercity\_set dest(src);
51. ~Univercity\_set();
52. };
53. #endif

Приложение Б.3 – файл set\_destructors.h – файл, содержащий реализацию всех деструкторов, используемых для множества.

1. #ifndef SET\_DESTRUCTORS
2. #define SET\_DESTRUCTORS
3. #include "set\_definitions.h"
4. template <typename T>
5. Node<T>::~Node() {
6. if (next) {
7. Node<T>\* tmp = owner->find\_first\_hash(Univercity\_set<T>::hash(value));
8. if (tmp == this)
9. owner->hash\_table[Univercity\_set<T>::hash(value)] = next;
10. else {
11. while (tmp && tmp->next != this) tmp = tmp->next;
12. if (tmp) tmp->next = this->next;
13. }
14. }
15. }
16. template <typename T>
17. Univercity\_set<T>::~Univercity\_set() {
18. // clear();
19. }
20. #endif

Приложение Б.4 – файл set\_hash.h – файл, содержащий реализацию метода, благодаря которому можно получить хеш-код для элемента.

1. #ifndef SET\_HASH
2. #define SET\_HASH
3. #include "set\_definitions.h"
4. template <typename T>
5. std::size\_t Univercity\_set<T>::hash(T value) {
6. return value.hash() % (MAX\_HASH);
7. }
8. template <>
9. std::size\_t Univercity\_set<int>::hash(int value) {
10. return (std::size\_t)(value ^ HASH\_MASK\_FOR\_INT) % (MAX\_HASH);
11. }
12. template <>
13. std::size\_t Univercity\_set<double>::hash(double value) {
14. return ((unsigned int)value ^ HASH\_MASK\_FOR\_INT) % (MAX\_HASH);
15. }
16. template <>
17. std::size\_t Univercity\_set<std::string>::hash(std::string value) {
18. return ((sizeof(value) + (value[0] + value[value.length() - 1]) / 2) % (MAX\_HASH));
19. }
20. template <typename T>
21. Node<T>\* Univercity\_set<T>::find\_first\_hash(std::size\_t hash) {
22. return hash\_table[hash];
23. }
24. #endif

Приложение Б.5 – файл set\_operations.h – файл, содержащий реализацию методов, которые зависят от экземпляра класса и не являются методами, которые подходят под описание прочих файлов.

1. #ifndef SET\_OPERATION
2. #define SET\_OPERATION
3. #include "set\_definitions.h"
4. #include "set\_destructors.h"
5. template <typename T>
6. void Univercity\_set<T>::add(T value) {
7. int index = Univercity\_set<T>::hash(value);
8. Node<T>\* tmp = hash\_table[index];
9. while (tmp) {
10. if (tmp->value == value) {
11. return;
12. }
13. tmp = tmp->next;
14. }
15. tmp = hash\_table[index];
16. if (!hash\_table[index])
17. hash\_table[index] = new Node<T>(value, this);
18. else {
19. while (tmp->next) {
20. tmp = tmp->next;
21. }
22. tmp->next = new Node<T>(value, this);
23. }
24. len++;
25. }
26. template <typename T>
27. void Univercity\_set<T>::del(T value) {
28. int index = Univercity\_set<T>::hash(value);
29. if (hash\_table[index]) {
30. Node<T>\* tmp = hash\_table[index];
31. Node<T>\* prev = nullptr;
32. while (tmp && !(tmp->value == value)) {
33. prev = tmp;
34. tmp = tmp->next;
35. }
36. if (tmp) {
37. if (prev) {
38. prev->next = tmp->next;
39. } else {
40. hash\_table[index] = tmp->next;
41. }
42. delete tmp;
43. len--;
44. }
45. }
46. }
47. template <>
48. void Univercity\_set<double>::del(double value) {
49. int index = Univercity\_set<double>::hash(value);
50. if (hash\_table[index]) {
51. Node<double>\* tmp = hash\_table[index];
52. Node<double>\* prev = nullptr;
53. while (tmp && fabs(tmp->value - value) >= 1e-6 ) {
54. prev = tmp;
55. tmp = tmp->next;
56. }
57. if (tmp) {
58. if (prev) {
59. prev->next = tmp->next;
60. } else {
61. hash\_table[index] = tmp->next;
62. }
63. delete tmp;
64. len--;
65. }
66. }
67. }
68. template <typename T>
69. void Univercity\_set<T>::clear() {
70. for (int i=0;i<MAX\_HASH;i++) {
71. if (hash\_table[i]) {
72. delete hash\_table[i];
73. hash\_table[i] = nullptr;
74. }
75. }
76. len = 0;
77. }
78. template <typename T>
79. bool Univercity\_set<T>::in(T value) {
80. std::size\_t index = Univercity\_set<T>::hash(value);
81. if (!hash\_table[index]) return false;
82. Node<T> \* tmp = hash\_table[index];
83. while (tmp) {
84. if (tmp->value == value) return true;
85. tmp = tmp->next;
86. }
87. return false;
88. }
89. template <typename T>
90. Univercity\_set<T> Univercity\_set<T>::union\_set(Univercity\_set<T> another) {
91. Univercity\_set<T> res = another;
92. for (int i = 0; i < MAX\_HASH; i++) {
93. Node<T> \* tmp = hash\_table[i];
94. while (tmp) {
95. res.add(tmp->value);
96. tmp = tmp->next;
97. }
98. }
99. return res;
100. }
101. template <typename T>
102. Univercity\_set<T> Univercity\_set<T>::intersection\_set(Univercity\_set<T> another) {
103. Univercity\_set<T> res;
105. for (int i = 0; i < MAX\_HASH; i++) {
106. Node<T> \* tmp = hash\_table[i];
107. while (tmp) {
108. if (another.in(tmp->value)) res.add(tmp->value);
109. tmp = tmp->next;
110. }
111. }
112. return res;
113. }
114. template <typename T>
115. Univercity\_set<T> Univercity\_set<T>::difference\_set(Univercity\_set<T> another) {
116. Univercity\_set<T> res;
117. for (int i = 0; i < MAX\_HASH; i++) {
118. Node<T> \* tmp = hash\_table[i];
119. while (tmp) {
120. if (!another.in(tmp->value)) res.add(tmp->value);
121. tmp = tmp->next;
122. }
123. }
124. return res;
125. }
126. #endif

Приложение Б.6 – файл set\_operators.h – файл, содержащий реализацию всех операторов, используемых для реализации множества.

1. #ifndef SET\_OPERATORS
2. #define SET\_OPERATORS
3. #include "set\_definitions.h"
4. template <typename T>
5. bool Univercity\_set<T>::operator==(Univercity\_set<T> &value) {
6. if (len != value.len) return false;
7. for (int i = 0; i < MAX\_HASH; i++) {
8. Node<T> \* tmp = hash\_table[i];
9. while (tmp) {
10. if (!value.in(tmp->value)) return false;
11. tmp = tmp->next;
12. }
14. }
15. return true;
16. }
17. #endif

Приложение Б.7 – файл set\_out.h – файл, содержащий реализацию методов, которые позволяют вывести множество в консоль.

1. #ifndef SET\_OUT
2. #define SET\_OUT
3. #include "set\_definitions.h"
4. #include <iostream>
5. template <typename T>
6. void Univercity\_set<T>::out\_value(T value) {
7. Node<T> \* tmp = hash\_table[Univercity\_set<T>::hash(value)];
8. while (tmp == nullptr && !(tmp->value == value))
9. tmp=tmp->next;
10. if (tmp)
11. tmp->value.out();
12. else
13. std::cout << "(null)";
14. }
15. template <>
16. void Univercity\_set<int>::out\_value(int value) {
17. std::cout << value;
18. }
19. template <>
20. void Univercity\_set<double>::out\_value(double value) {
21. std::cout << value;
22. }
23. template <>
24. void Univercity\_set<std::string>::out\_value(std::string value) {
25. std::cout << value;
26. }
27. template <typename T>
28. void Univercity\_set<T>::out() {
29. for (int i = 0; i < MAX\_HASH; i++) {
30. if (hash\_table[i]) {
31. Node<T> \* tmp = hash\_table[i];
32. while (tmp) {
33. out\_value(tmp->value);
34. std::cout << " ";
35. tmp = tmp->next;
36. }
37. }
38. }
39. }
40. template <>
41. void Univercity\_set<int>::out() {
42. for (int i = 0; i < MAX\_HASH; i++) {
43. Node<int> \* tmp = hash\_table[i];
44. while (tmp) {
45. out\_value(tmp->value);
46. std::cout << " ";
47. tmp = tmp->next;
48. }
49. }
50. std::cout << std::endl;
51. }
52. template <>
53. void Univercity\_set<double>::out() {
54. for (int i = 0; i < MAX\_HASH; i++) {
55. if (hash\_table[i]) {
56. Node<double> \* tmp = hash\_table[i];
57. while (tmp) {
58. out\_value(tmp->value);
59. std::cout << " ";
60. tmp = tmp->next;
61. }
62. }
63. }
64. std::cout << std::endl;
65. }
66. template <>
67. void Univercity\_set<std::string>::out() {
68. for (int i = 0; i < MAX\_HASH; i++) {
69. if (hash\_table[i]) {
70. Node<std::string> \* tmp = hash\_table[i];
71. while (tmp) {
72. out\_value(tmp->value);
73. std::cout << " ";
74. tmp = tmp->next;
75. }
76. }
77. }
78. std::cout << std::endl;
79. }
80. #endif

Приложение Б.8 – файл set.h – файл, подключающий остальные файлы.

1. #ifndef UNIVERCITY\_SET\_H
2. #define UNIVERCITY\_SET\_H
3. #include "set\_definitions.h"
4. #include "set\_constructors.h"
5. #include "set\_hash.h"
6. #include "set\_out.h"
7. #include "set\_operations.h"
8. #include "set\_operators.h"
9. #endif

Приложение Б.9 – файл test.h – файл, содержащий реализацию структуры, которую можно хранить в множестве и макроопределений для тестирования.

1. #ifndef TEST\_STRUCT
2. #define TEST\_STRUCT
3. #include <iostream>
4. struct Coordinates {
5. int x, y, z;
6. Coordinates() : x(0), y(0), z(0){};
7. Coordinates(int x\_, int y\_, int z\_) : x(x\_), y(y\_), z(z\_){};
8. std::size\_t hash() {return (x \* y \* z);};
9. void out() {
10. std::cout << x << " " << y << " " << z << std::endl;
11. }
12. bool operator == (const Coordinates & cmp) {
13. return (x == cmp.x && y == cmp.y && z == cmp.z);
14. }
15. };
16. #define ASSERT\_HINT(val, hint, ntest) \
17. if (val == false) std::cout << "Error in "\_\_FILE\_\_ << " file, "<<\_\_LINE\_\_ \
18. << " line, " << \_\_FUNCTION\_\_ << " func, "<< hint; \
19. else std::cout << "TEST " << ntest << " success!" << std::endl;
20. #define ASSERT(val, ntest) ASSERT\_HINT(val, NULL, ntest)
21. #endif

Приложение Б.10 – файл test.cpp – файл, благодаря которому производилось тестирование работоспособности класса.

1. #include <iostream>
2. #include <string>
3. #include "set.h"
4. #include "test.h"
5. static void init\_set(Univercity\_set<int> \* int\_set, Univercity\_set<double> \* double\_set,
6. Univercity\_set<std::string> \* string\_set, Univercity\_set<Coordinates> \* coordinates\_set);
7. static void out\_set(Univercity\_set<int> \* int\_set, Univercity\_set<double> \* double\_set,
8. Univercity\_set<std::string> \* string\_set, Univercity\_set<Coordinates> \* coordinates\_set);
9. static void del\_one\_elem\_set(Univercity\_set<int> \* int\_set, Univercity\_set<double> \* double\_set,
10. Univercity\_set<std::string> \* string\_set, Univercity\_set<Coordinates> \* coordinates\_set);
11. int main() {
12. Univercity\_set<int> int\_set;
13. Univercity\_set<double> double\_set;
14. Univercity\_set<std::string> string\_set;
15. Univercity\_set<Coordinates> coordinates\_set;
16. // out empty set
17. out\_set(&int\_set, &double\_set, &string\_set, &coordinates\_set);
18. // simple delete
19. init\_set(&int\_set, &double\_set, &string\_set, &coordinates\_set);
20. std::cout << "\n\nbefore delete" << std::endl;
21. out\_set(&int\_set, &double\_set, &string\_set, &coordinates\_set);
22. std::cout << "\n\nafter delete" << std::endl;
23. del\_one\_elem\_set(&int\_set, &double\_set, &string\_set, &coordinates\_set);
24. out\_set(&int\_set, &double\_set, &string\_set, &coordinates\_set);
26. // add same elem
28. std::cout << int\_set.len << std::endl;
29. ASSERT(int\_set.len == 2, 1);
30. int\_set.add(2);
31. int\_set.add(2);
32. int\_set.add(3);
33. int\_set.add(2);
34. ASSERT(int\_set.len == 2, 1);
35. // add with same hash
36. int\_set.add(1002);
37. int\_set.add(1050);
38. int\_set.add(2098);
39. int\_set.add(3146);
40. int\_set.add(4194);
41. ASSERT(int\_set.len == 7, 2);
42. // del same hash
43. int\_set.del(2098); // middle
44. int\_set.del(1002); // start
45. int\_set.del(4194); // end
46. ASSERT(int\_set.len == 4, 3);
47. // del unreal elements
48. int\_set.del(2098); // middle
49. int\_set.del(1002); // start
50. int\_set.del(4194); // end
51. ASSERT(int\_set.len == 4, 4);
52. // del all elements
53. int\_set.del(3);
54. int\_set.del(2);
55. int\_set.del(3146);
56. int\_set.del(1050);
57. ASSERT(int\_set.empty(), 5);
58. int\_set.add(3);
59. int\_set.add(2);
60. int\_set.add(1002);
61. int\_set.add(3146);
62. int\_set.clear();
63. ASSERT(int\_set.len == 0, 6);
64. // copy test
65. Univercity\_set<int> test\_int\_set = int\_set;
66. ASSERT(test\_int\_set.len == 0, 7);
67. int\_set.add(3);
68. int\_set.add(2);
69. int\_set.add(1002);
70. int\_set.add(3146);
71. test\_int\_set = int\_set;
72. ASSERT(test\_int\_set.len == 4, 8);
74. ASSERT(int\_set.in(1002) && int\_set.in(3146) && int\_set.in(2) && int\_set.in(3), 9);
75. ASSERT(int\_set == test\_int\_set, 10);
76. int\_set.add(1);
77. int\_set.add(1001);
78. int\_set.add(9);
79. test\_int\_set.add(666);
80. int arr\_int[4] = {3, 2, 1002, 3146};
81. Univercity\_set<int> assert\_int;
82. for (auto & a: arr\_int) assert\_int.add(a);
83. ASSERT(int\_set.intersection\_set(test\_int\_set) == assert\_int, 11);
84. assert\_int.clear();
85. int arr\_int\_1[3] = {9, 1, 1001};
86. for (auto & a: arr\_int\_1) assert\_int.add(a);
87. ASSERT(int\_set.difference\_set(test\_int\_set) == assert\_int, 12);
88. int arr\_int\_2[8] = {9, 3, 2, 1002, 1, 1001, 3146, 666};
89. assert\_int.clear();
90. for (auto & a: arr\_int\_2) assert\_int.add(a);
91. ASSERT(int\_set.union\_set(test\_int\_set) == assert\_int, 13);
92. return 0;
93. }
94. static void init\_set(Univercity\_set<int> \* int\_set, Univercity\_set<double> \* double\_set,
95. Univercity\_set<std::string> \* string\_set, Univercity\_set<Coordinates> \* coordinates\_set) {
96. int\_set->add(1);
97. int\_set->add(2);
98. int\_set->add(3);
99. double\_set->add(1.1);
100. double\_set->add(2.1);
101. double\_set->add(3.1);
103. string\_set->add("asd");
104. string\_set->add("dsa");
105. string\_set->add("zxc");
106. coordinates\_set->add(Coordinates(1, 2, 3));
107. coordinates\_set->add(Coordinates(9, 7, 5));
108. coordinates\_set->add(Coordinates(2, 3, 4));
109. coordinates\_set->add(Coordinates(2000000000, 3456464, 475));
110. }
111. static void out\_set(Univercity\_set<int> \* int\_set, Univercity\_set<double> \* double\_set,
112. Univercity\_set<std::string> \* string\_set, Univercity\_set<Coordinates> \* coordinates\_set) {
113. int\_set->out();
114. double\_set->out();
115. string\_set->out();
116. coordinates\_set->out();
117. }
118. static void del\_one\_elem\_set(Univercity\_set<int> \* int\_set, Univercity\_set<double> \* double\_set,
119. Univercity\_set<std::string> \* string\_set, Univercity\_set<Coordinates> \* coordinates\_set) {
120. coordinates\_set->del(Coordinates(2000000000, 3456464, 475));
121. double\_set->del(2.1);
122. string\_set->del("asd");
123. int\_set->del(1);
124. }

Приложение Б.11 – файл lab1.cpp – файл, содержайщий реализацию различных способов генерации последовательностей.

1. #include <iostream>
2. #include <ctime>
3. #include <cstdlib>
4. #include "lab1.h"
5. #include <cmath>
6. void int\_array::output() {
7. for (int i=0;i<size;i++)
8. printf("%d ", arr[i]);
9. printf("\n");
10. }
11. void int\_array::gen\_rand(int min\_val, int max\_val) {
12. for (int i=0;i<size;i++) {
13. arr[i] = rand() % (max\_val-min\_val+1) + min\_val;
14. }
15. }
16. // max\_val no control
17. void int\_array::gen\_up(int min\_val, int max\_val, int step) {
18. int val = min\_val;
19. int i = 0;
20. for (i=0;i<size;i++) {
21. arr[i] = val;
22. if (val+step <= max\_val)
23. val += step;
24. else
25. val = max\_val;
26. }
27. }
28. void int\_array::gen\_down(int min\_val, int max\_val, int step) {
29. int val = max\_val;
30. int i = 0;
31. for (i=0;i<size;i++) {
32. arr[i] = val;
34. if (val-step >= min\_val)
35. val -= step;
36. else
37. val = min\_val;
38. }
39. }
40. void int\_array::sin\_gen(int min\_val, int max\_val) {
41. int mid, i = 0, n = size, j = 0, val = max\_val, step=(max\_val-min\_val)/interval;
42. mid = interval / 2;
43. while (n - interval >= 0) {
44. for (i=0+j;i<mid+j;i++) {
45. arr[i] = val;
46. if (val-step >= min\_val)
47. val -= step;
48. else
49. val = min\_val;
50. }
51. for (i=mid+j;i<interval+j;i++) {
52. arr[i] = val;
53. if (val+step <= max\_val)
54. val += step;
55. else
56. val = max\_val;
57. }
58. n -= interval;
59. j += interval;
60. val = max\_val;
61. }
62. }
63. //a1 < a2 > a3 < … > an-1 < an
64. void int\_array::sawtooth\_gen(int min\_val, int max\_val) {
65. int i=0, j=0, val=min\_val, n=size, step=(max\_val-min\_val)/interval;
66. while (n - interval >= 0) {
67. for (i=0+j;i<interval+j;i++) {
68. arr[i] = val;
69. if (val+step <= max\_val)
70. val += step;
71. else
72. val = max\_val;
73. }
74. val = min\_val;
75. n -= interval;
76. j += interval;
77. }
78. }
79. void int\_array::step\_gen(int min\_val, int max\_val) {
80. int range = max\_val - min\_val, i=0, j=0, n=size, step = range / (n / interval) / 2, mn=step+min\_val, mx = 2\*step+min\_val;
81. while (n - interval >= 0) {
82. for (i=0+j;i<interval+j;i++) {
83. arr[i] = rand() % (mx-mn+1) + mn;
84. }
85. n -= interval;
86. j += interval;
87. mn += 2 \* step;
88. mx += 2 \* step;
89. }
90. }
91. void int\_array::kvazi\_gen(int min\_val, int max\_val) {
92. int i=0, j=0, n=size, val = min\_val;
93. int step = (max\_val - min\_val) / (n / interval) / 2, mn=step+min\_val, mx = 2\*step+min\_val;
94. while (n - interval >= 0) {
95. for (i=0+j;i<interval+j;i++) {
96. arr[i] = val;
98. if (val+step <= mx)
99. val += (step/interval);
100. else
101. val = mx;
102. }
103. i--;
104. inversion\_gen(j);
105. n -= interval;
106. j += interval;
107. mn += 2 \* step;
108. mx += 2 \* step;
109. }
110. }
111. void int\_array::inversion\_gen(int start) {
112. int inversions = 0, swapped, temp;
113. int mx\_index, mn\_index;
114. inversions = rand() % (interval / 2 + 1);
115. switch (inversions) {
116. case 1:
117. mn\_index = 0;
118. mx\_index = interval-1;
119. swapped = rand() % (mx\_index+1) + mn\_index;
120. // свап влево или вправо
121. if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index) {
122. swap\_arr(arr, start+swapped, start+swapped - 1, 1);
123. }
124. else {
125. swap\_arr(arr, start+swapped, start+swapped + 1, 1);
126. }
127. break;
128. case 2:
129. mn\_index = 0;
130. mx\_index = (interval/2)-1;
131. swapped = rand() % (mx\_index+1) + mn\_index;
132. if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index)
133. swap\_arr(arr, start+swapped, start+swapped - 1, 1);
134. else
135. swap\_arr(arr, start+swapped, start+swapped + 1, 1);
136. mn\_index = interval/2;
137. mx\_index = interval/2;
138. swapped = rand() % (mx\_index) + mn\_index;
139. if ((rand() % 2 && swapped != mn\_index) || swapped == mn\_index+mx\_index-1)
140. swap\_arr(arr, start+swapped, start+swapped - 1, 1);
141. else
142. swap\_arr(arr, start+swapped, start+swapped + 1, 1);
143. break;
144. case 3:
145. mn\_index = 0;
146. mx\_index = interval/3;
147. swapped = rand() % mx\_index;
148. if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1)
149. swap\_arr(arr, start+swapped, start+swapped - 1, 1);
150. else
151. swap\_arr(arr, start+swapped, start+swapped + 1, 1);
152. mn\_index = interval/3;
153. swapped = rand() % (mx\_index) + mn\_index;
154. if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index + mn\_index - 1)
155. swap\_arr(arr, start+swapped, start+swapped - 1, 1);
156. else
157. swap\_arr(arr, start+swapped, start+swapped + 1, 1);
158. mn\_index = interval/3 \* 2;
159. swapped = rand() % (mx\_index) + mn\_index;
160. if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index + mn\_index - 1)
161. swap\_arr(arr, start+swapped, start+swapped - 1, 1);
162. else
163. swap\_arr(arr, start+swapped, start+swapped + 1, 1);
164. break;
165. case 4:
166. temp = rand() % 3; // в каком месте встваим на 2 позиции
167. mn\_index = 0;
168. mx\_index = interval/3;
169. swapped = rand() % mx\_index;
170. if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))
171. swap\_arr(arr, start+swapped, start+swapped - 1, 1);
172. else if (temp == 0) {
173. // mx elem
174. swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 1);
175. swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 1);
176. }
177. else
178. swap\_arr(arr, start+swapped, start+swapped + 1, 1);
179. mn\_index = interval/3 \* 2;
180. swapped = rand() % (mx\_index) + mn\_index;
181. if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))
182. swap\_arr(arr, start+swapped, start+swapped - 1, 1);
183. else if (temp == 1) {
184. // mx elem
185. swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 1);
186. swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 1);
187. }
188. else
189. swap\_arr(arr, start+swapped, start+swapped + 1, 1);
190. mn\_index = 0;
191. mx\_index = interval/3;
192. swapped = rand() % mx\_index;
193. if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))
194. swap\_arr(arr, start+swapped, start+swapped - 1, 1);
195. else if (temp == 2) {
196. // mx elem
197. swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 1);
198. swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 1);
199. }
200. else
201. swap\_arr(arr, start+swapped, start+swapped + 1, 1);
202. break;
203. case 5:
204. temp = rand() % 3; // в каком месте не встваим на 2 позиции
205. mn\_index = 0;
206. mx\_index = interval/3;
207. swapped = rand() % mx\_index;
208. if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))
209. swap\_arr(arr, start+swapped, start+swapped - 1, 1);
210. else if (temp != 0) {
211. // mx elem
212. swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 1);
213. swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 1);
214. }
215. else
216. swap\_arr(arr, start+swapped, start+swapped + 1, 1);
217. mn\_index = interval/3 \* 2;
218. swapped = rand() % (mx\_index) + mn\_index;
219. if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))
220. swap\_arr(arr, start+swapped, start+swapped - 1, 1);
221. else if (temp != 1) {
222. // mx elem
223. swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 1);
224. swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 1);
225. }
226. else
227. swap\_arr(arr, start+swapped, start+swapped + 1, 1);
228. mn\_index = 0;
229. mx\_index = interval/3;
230. swapped = rand() % mx\_index;
231. if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))
232. swap\_arr(arr, start+swapped, start+swapped - 1, 1);
233. else if (temp != 2) {
234. // mx elem
235. swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 1);
236. swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 1);
237. }
238. else
239. swap\_arr(arr, start+swapped, start+swapped + 1, 1);
240. break;
241. }
242. }
243. void double\_array::gen\_down(double min\_val, double max\_val, double step) {
244. double val = max\_val;
245. for (int i=0;i<size;i++) {
246. if (val > min\_val) {
247. arr[i] = val;
248. val -= step;
249. }
250. else {
251. arr[i] = min\_val;
252. }
253. }
254. }
255. void double\_array::gen\_up(double min\_val, double max\_val, double step) {
256. double val = min\_val;
257. for (int i=0;i<size;i++) {
258. if (val < max\_val) {
259. arr[i] = val;
260. val += step;
261. }
262. else {
263. arr[i] = max\_val;
264. }
265. }
266. }
267. void double\_array::gen\_rand(double min\_val, double max\_val) {
268. int dif = max\_val - min\_val;
269. double temp = 0;
270. if (dif > 0) {
271. for (int i=0;i<size;i++) {
272. arr[i] = (double)(rand() % dif) + min\_val;
274. temp = rand() % 500 + 500; // 0.5 - 0.999
275. while (temp > 1) {
276. temp /=10;
277. }
278. arr[i] \*= temp;
279. if (arr[i] < min\_val) arr[i] = min\_val;
280. }
281. }
282. else if (dif == 0) {
283. int count = 0;
284. while (min\_val < 1) {
285. min\_val \*= 10;
286. max\_val \*= 10;
287. count++;
288. }
289. for (int i=0;i<size;i++) {
290. arr[i] = (double)(rand() % ((int)max\_val - (int)min\_val + 1) + (int)min\_val) / pow(10, count);
291. temp = rand() % 1000 + 9000;
292. while (temp > 1) {
293. temp /=10;
294. }
295. arr[i] \*= temp;
296. if (arr[i] < min\_val/pow(10, count)) arr[i] = min\_val/pow(10, count);
297. if (arr[i] > max\_val/pow(10, count)) arr[i] = max\_val/pow(10, count);
298. }
300. }
301. }
302. void double\_array::sin\_gen(double min\_val, double max\_val) {
303. int mid, i = 0, n = size, j = 0;
304. double val = max\_val, step=(max\_val-min\_val)/interval;
305. mid = interval / 2;
306. while (n - interval >= 0) {
307. for (i=0+j;i<mid+j;i++) {
308. arr[i] = val;
309. if (val-step >= min\_val)
310. val -= step;
311. else
312. val = min\_val;
313. }
314. for (i=mid+j;i<interval+j;i++) {
315. arr[i] = val;
316. if (val+step <= max\_val)
317. val += step;
318. else
319. val = max\_val;
320. }
321. n -= interval;
322. j += interval;
323. val = max\_val;
324. }
325. }
326. void double\_array::step\_gen(double min\_val, double max\_val) {
327. int j=0, n=size, dif = 0;
328. double range = max\_val - min\_val, step = range / (n / interval) / 2, mn=step+min\_val, mx = 2\*step+min\_val, temp = 0;
330. while (n - interval >= 0) {
331. dif = (int)mx - (int)mn;
332. // printf("%lg, %lg, %d\n", mn, mx, dif);
333. if (dif > 0) {
334. for (int i=0+j;i<interval+j;i++) {
335. arr[i] = (double)(rand() % (dif+1)) + mn;
337. temp = rand() % 1000 + 9000; // 0.9 - 0.999
338. while (temp > 1) {
339. temp /=10;
340. }
341. arr[i] \*= temp;
342. if (arr[i] < mn) arr[i] = mn;
343. if (arr[i] > mx) arr[i] = mx;
344. }
345. }
346. else if (dif == 0) {
347. int count = 0;
348. while (mn < 1) {
349. mn \*= 10;
350. mx \*= 10;
351. count++;
352. }
353. for (int i=0+j;i<interval+j;i++) {
354. arr[i] = (double)(rand() % ((int)mx - (int)mn + 1) + (int)mn) / pow(10, count);
355. temp = rand() % 1000 + 9000;
356. while (temp > 1) {
357. temp /=10;
358. }
359. arr[i] \*= temp;
360. if (arr[i] < mn/pow(10, count)) arr[i] = mn/pow(10, count);
361. if (arr[i] > mx/pow(10, count)) arr[i] = mx/pow(10, count);
362. }
363. mn /= pow(10, count);
364. mx /= pow(10, count);
365. }
366. n -= interval;
367. j += interval;
368. mn += 2 \* step;
369. mx += 2 \* step;
370. }
371. }
372. void double\_array::sawtooth\_gen(double min\_val, double max\_val) {
373. int i=0, j=0, n=size;
374. double val = min\_val, step=(max\_val-min\_val)/interval;
375. while (n - interval >= 0) {
376. for (i=0+j;i<interval+j;i++) {
377. arr[i] = val;
378. if (val+step <= max\_val)
379. val += step;
380. else
381. val = max\_val;
382. }
383. val = min\_val;
384. n -= interval;
385. j += interval;
386. }
387. }
388. void double\_array::kvazi\_gen(double min\_val, double max\_val) {
389. int i=0, j=0, n=size;
390. double step = (max\_val - min\_val) / (n / interval) / 2, mn=step+min\_val, mx = 2\*step+min\_val, val = min\_val;
391. while (n - interval >= 0) {
392. for (i=0+j;i<interval+j;i++) {
393. arr[i] = val;
395. if (val+step <= mx)
396. val += (step/interval);
397. else
398. val = mx;
399. }
400. i--;
401. inversion\_gen(j);
402. n -= interval;
403. j += interval;
404. mn += 2 \* step;
405. mx += 2 \* step;
406. }
407. }
408. void double\_array::inversion\_gen(int start) {
409. int inversions = 0, swapped, temp;
410. int mx\_index, mn\_index;
411. inversions = rand() % (interval / 2 + 1);
412. switch (inversions) {
413. case 1:
414. mn\_index = 0;
415. mx\_index = interval-1;
416. swapped = rand() % (mx\_index+1) + mn\_index;
417. // свап влево или вправо
418. if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index) {
419. swap\_arr(arr, start+swapped, start+swapped - 1, 0);
420. }
421. else {
422. swap\_arr(arr, start+swapped, start+swapped + 1, 0);
423. }
424. break;
425. case 2:
426. mn\_index = 0;
427. mx\_index = (interval/2)-1;
428. swapped = rand() % (mx\_index+1) + mn\_index;
429. if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index)
430. swap\_arr(arr, start+swapped, start+swapped - 1, 0);
431. else
432. swap\_arr(arr, start+swapped, start+swapped + 1, 0);
433. mn\_index = interval/2;
434. mx\_index = interval/2;
435. swapped = rand() % (mx\_index) + mn\_index;
436. if ((rand() % 2 && swapped != mn\_index) || swapped == mn\_index+mx\_index-1)
437. swap\_arr(arr, start+swapped, start+swapped - 1, 0);
438. else
439. swap\_arr(arr, start+swapped, start+swapped + 1, 0);
440. break;
441. case 3:
442. mn\_index = 0;
443. mx\_index = interval/3;
444. swapped = rand() % mx\_index;
445. if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1)
446. swap\_arr(arr, start+swapped, start+swapped - 1, 0);
447. else
448. swap\_arr(arr, start+swapped, start+swapped + 1, 0);
449. mn\_index = interval/3;
450. swapped = rand() % (mx\_index) + mn\_index;
451. if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index + mn\_index - 1)
452. swap\_arr(arr, start+swapped, start+swapped - 1, 0);
453. else
454. swap\_arr(arr, start+swapped, start+swapped + 1, 0);
455. mn\_index = interval/3 \* 2;
456. swapped = rand() % (mx\_index) + mn\_index;
457. if ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index + mn\_index - 1)
458. swap\_arr(arr, start+swapped, start+swapped - 1, 0);
459. else
460. swap\_arr(arr, start+swapped, start+swapped + 1, 0);
461. break;
462. case 4:
463. temp = rand() % 3; // в каком месте встваим на 2 позиции
464. mn\_index = 0;
465. mx\_index = interval/3;
466. swapped = rand() % mx\_index;
467. if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))
468. swap\_arr(arr, start+swapped, start+swapped - 1, 0);
469. else if (temp == 0) {
470. // mx elem
471. swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 0);
472. swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 0);
473. }
474. else
475. swap\_arr(arr, start+swapped, start+swapped + 1, 0);
476. mn\_index = interval/3 \* 2;
477. swapped = rand() % (mx\_index) + mn\_index;
478. if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))
479. swap\_arr(arr, start+swapped, start+swapped - 1, 0);
480. else if (temp == 1) {
481. // mx elem
482. swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 0);
483. swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 0);
484. }
485. else
486. swap\_arr(arr, start+swapped, start+swapped + 1, 0);
487. mn\_index = 0;
488. mx\_index = interval/3;
489. swapped = rand() % mx\_index;
490. if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))
491. swap\_arr(arr, start+swapped, start+swapped - 1, 0);
492. else if (temp == 2) {
493. // mx elem
494. swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 0);
495. swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 0);
496. }
497. else
498. swap\_arr(arr, start+swapped, start+swapped + 1, 0);
499. break;
500. case 5:
501. temp = rand() % 3; // в каком месте не встваим на 2 позиции
502. mn\_index = 0;
503. mx\_index = interval/3;
504. swapped = rand() % mx\_index;
505. if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))
506. swap\_arr(arr, start+swapped, start+swapped - 1, 0);
507. else if (temp != 0) {
508. // mx elem
509. swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 0);
510. swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 0);
511. }
512. else
513. swap\_arr(arr, start+swapped, start+swapped + 1, 0);
514. mn\_index = interval/3 \* 2;
515. swapped = rand() % (mx\_index) + mn\_index;
516. if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))
517. swap\_arr(arr, start+swapped, start+swapped - 1, 0);
518. else if (temp != 1) {
519. // mx elem
520. swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 0);
521. swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 0);
522. }
523. else
524. swap\_arr(arr, start+swapped, start+swapped + 1, 0);
525. mn\_index = 0;
526. mx\_index = interval/3;
527. swapped = rand() % mx\_index;
528. if (temp != 0 && ((rand() % 2 && swapped != mn\_index) || swapped == mx\_index - 1))
529. swap\_arr(arr, start+swapped, start+swapped - 1, 0);
530. else if (temp != 2) {
531. // mx elem
532. swap\_arr(arr, mn\_index, mn\_index+mx\_index-1, 0);
533. swap\_arr(arr, mn\_index+1, mn\_index+mx\_index-1, 0);
534. }
535. else
536. swap\_arr(arr, start+swapped, start+swapped + 1, 0);
537. break;
538. }
539. }
540. // 0-2 2-0 элемента переставляем на 1 позицию (свап)
541. // 3 выбираем 3 случайных числа 1-8 и вставляем
542. // 4 - выбираем 2 элемента 1-2+7-8 и вставляем, дальше 1 элемент 3-6 и вставляем либо вправо либо влево на 2
543. // 5 - выбираем 1 элемент 1-9
544. // можно вставлять вместо свапа - на n элементов переставили, получили n инверсий
545. // нужно переставить 3 элемента
546. // 0 - double\*, 1 - int\*
547. void swap\_arr(void \* arr\_, int i, int j, int doub\_or\_int) {
548. int temp1 = 0;
549. double temp2;
550. if (doub\_or\_int) {
551. int \* arr = (int\*)arr\_;
552. temp1 = arr[i];
553. arr[i] = arr[j];
554. arr[j] = temp1;
555. }
556. else {
557. double \* arr = (double\*)arr\_;
558. temp2 = arr[i];
559. arr[i] = arr[j];
560. arr[j] = temp2;
561. }
562. }
563. void double\_array::output() {
564. for (int i=0;i<size;i++)
565. printf("%lg ", arr[i]);
566. printf("\n");
567. }
568. void start\_gen(double\_array \* gen\_1, int\_array \* gen\_2, int min\_val\_int, int max\_val\_int,
569. double min\_val\_double, double max\_val\_double, int int\_step, double double\_step) {
570. cout << "Int generation start\n\n";
571. cout << "down generation\n";
572. cout << "\n\n" << int\_step <<"\n\n\n";
573. clock\_t start\_time = clock();
574. gen\_2->gen\_down(min\_val\_int, max\_val\_int, int\_step);
575. cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " sec" << endl;
576. start\_time = clock();
577. gen\_2->output();
578. cout << "up generation\n";
579. gen\_2->gen\_up(min\_val\_int, max\_val\_int, int\_step);
580. cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " sec" << endl;
581. start\_time = clock();
582. gen\_2->output();
583. cout << "random generation\n";
584. gen\_2->gen\_rand(min\_val\_int, max\_val\_int);
585. cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " sec" << endl;
586. start\_time = clock();
587. gen\_2->output();
588. cout << "sawtooth generation\n";
589. gen\_2->sawtooth\_gen(min\_val\_int, max\_val\_int);
590. cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " sec" << endl;
591. start\_time = clock();
592. gen\_2->output();
593. cout << "sin generation\n";
594. gen\_2->sin\_gen(min\_val\_int, max\_val\_int);
595. cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " sec" << endl;
596. start\_time = clock();
597. gen\_2->output();
598. cout << "step generation\n";
599. gen\_2->step\_gen(min\_val\_int, max\_val\_int);
600. cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " sec" << endl;
601. start\_time = clock();
602. gen\_2->output();
603. cout << "kvazi generation\n";
604. gen\_2->kvazi\_gen(min\_val\_int, max\_val\_int);
605. cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " sec" << endl;
606. start\_time = clock();
607. gen\_2->output();
609. cout << "Double generation start\n\n";
610. cout << "down generation\n";
611. gen\_1->gen\_down(min\_val\_double, max\_val\_double, double\_step);
612. cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " sec" << endl;
613. start\_time = clock();
614. gen\_1->output();
615. cout << "up generation\n";
616. gen\_1->gen\_up(min\_val\_double, max\_val\_double, double\_step);
617. cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " sec" << endl;
618. start\_time = clock();
619. gen\_1->output();
620. cout << "random generation\n";
621. gen\_1->gen\_rand(min\_val\_double, max\_val\_double);
622. cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " sec" << endl;
623. start\_time = clock();
624. gen\_1->output();
625. cout << "sawtooth generation\n";
626. gen\_1->sawtooth\_gen(min\_val\_double, max\_val\_double);
627. cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " sec" << endl;
628. start\_time = clock();
629. gen\_1->output();
630. cout << "sin generation\n";// 1 2 3 4 5 6 7 8 9 10
631. cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " sec" << endl;
632. start\_time = clock();
633. gen\_1->output();
634. cout << "step generation\n";
635. gen\_1->step\_gen(min\_val\_double, max\_val\_double);
636. cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " sec" << endl;
637. start\_time = clock();
638. gen\_1->output();
639. cout << "kvazi generation\n";
640. gen\_1->kvazi\_gen(min\_val\_double, max\_val\_double);
641. cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " sec" << endl;
642. start\_time = clock();
643. gen\_1->output();
644. }
645. int main(){
646. srand(time(NULL));
647. int n = 0;
648. printf("enter size of array\n");
649. scanf("%d", &n);
650. double\_array gen\_1(n);
651. int\_array gen\_2(n);
652. start\_gen(&gen\_1, &gen\_2, 0, 1000, 0.123, 9.23, 15, 0.21);
653. return 0;
654. }
655. /\*
656. Квази-упорядоченная последовательность – это последовательность, в которой имеется
657. ограниченное количество инверсий (количество элементов, для которых не
658. выполняется отношение порядка) и оно много меньше размера последовательности.
659. \*/

Приложение Б.12 – файл lab1.h – файл, в котором находятся определения классов и функций, используемых в файле lab1.cpp.

1. #define interval 10
2. using namespace std;
3. void swap\_arr(void \* arr, int i, int j, int doub\_or\_int);
4. class double\_array {
5. private:
6. int size;
7. double \* arr;
8. void inversion\_gen(int start);
9. public:
10. void output();
11. void gen\_rand(double min\_val, double max\_val);
12. void gen\_up(double min\_val, double max\_val, double step);
13. void gen\_down(double min\_val, double max\_val, double step);
14. void step\_gen(double min\_val, double max\_val);
15. void sin\_gen(double min\_val, double max\_val);
16. void sawtooth\_gen(double min\_val, double max\_val);
17. void kvazi\_gen(double min\_val, double max\_val);
18. double\_array(int n) {
19. arr = new double[n];
20. size = n;
21. }
22. ~double\_array() {
23. delete arr;
24. }
25. };
26. class int\_array {
27. private:
28. int \* arr;
29. int size;
30. void inversion\_gen(int start);
31. public:
32. void output();
33. void gen\_rand(int min\_val, int max\_val);
34. void gen\_up(int min\_val, int max\_val, int step);
35. void gen\_down(int min\_val, int max\_val, int step);
36. void sin\_gen(int min\_val, int max\_val);
37. //a1 < a2 > a3 < … > an-1 < an
38. void sawtooth\_gen(int min\_val, int max\_val);
39. void step\_gen(int min\_val, int max\_val);
40. void kvazi\_gen(int min\_val, int max\_val);
41. int\_array(int n) {
42. arr = new int[n];
43. for (int i=0;i<n;i++) arr[i] = 0;
44. size = n;
45. }
46. ~int\_array() {
47. delete arr;
48. }
49. };

Приложение Б.13 – модуль для замера времени работы множества, реализованного в данной курсовой работе и множества, реализованного в STL.

1. #include "set.h"
2. #include "gen.h"
3. #include <unordered\_set>
4. void test\_time(int arr\_size, int step);
5. int main(int argc, char \*argv[]) {
6. setlocale(LC\_ALL, "ru\_RU.UTF-8");
7. /\*
8. rand gen
9. int arr\_sizes[6] = {10000000, 30000000, 50000000, 70000000, 90000000, 100000000};
10. std::cout << "my\tstl\n";
11. for (auto &a : arr\_sizes) {
12. test\_time(a, 0.93);
13. }
14. \*/
15. // up\_gen
16. // int arr\_sizes[6] = {1000000, 3000000, 5000000, 7000000, 9000000, 10000000};
17. // std::cout << "Генерация происходит с маленьким шагом\n";
18. // std::cout << "my\tstl\n";
19. // for (auto &a : arr\_sizes) {
20. // test\_time(a, 0.93);
21. // }
22. // std::cout << "Генерация происходит с большим шагом\n";
23. // std::cout << "my\tstl\n";
24. // for (auto &a : arr\_sizes) {
25. // test\_time(a, 10);
26. // }
27. // kvazi\_gen
28. // int arr\_sizes[6] = {1000000, 3000000, 5000000, 7000000, 9000000, 10000000};
29. // std::cout << "my\tstl\n";
30. // for (auto &a : arr\_sizes) {
31. // test\_time(a, 0.93);
32. // }
33. return 0;
34. }
35. void test\_time(int arr\_size, int step) {
36. Univercity\_set<double> double\_set;
37. double\_array test\_double(arr\_size);
38. // test\_double.gen\_up(1, 1e10, step);
39. // test\_double.kvazi\_gen(1, 1e10);
40. /\* rand gen
41. test\_double.gen\_rand(1, 1e10);
42. \*/
43. double \* arr = test\_double.get\_arr();
44. clock\_t start\_time = clock();
45. for (int i=0; i<arr\_size;i++) {
46. double\_set.add(arr[i]);
47. }
48. std::cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << " ";
49. unordered\_set<double> double\_stl\_set;
50. start\_time = clock();
51. for (int i=0; i<arr\_size;i++) {
52. double\_stl\_set.insert(arr[i]);
53. }
54. std::cout << (double)(clock()-start\_time) / CLOCKS\_PER\_SEC << std::endl;
55. }