МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Математическое обеспечение и применение ЭВМ»

Курсовая работа

по дисциплине «Программирование динамических структур данных»

на тему «Реализация структуры данных «Множество» с использованием контейнерных классов Multiset и queue»

ПГУ 09.03.04 - 04КР211.18 ПЗ

Направление подготовки - 09.03.04 Программная инженерия
Профиль подготовки - Программная инженерия

Выполнил студент _____ Мишихин В.Ю.

Группа 21ВП1

Руководитель
к.т.н., доцент _____ Самуйлов С.В.

Работа защищена с оценкой _____

Преподаватели _____

_____

Дата защиты _____

2023

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Математическое обеспечение и применение ЭВМ»

«УТВЕРЖДАЮ»
Зав. кафедрой _____ Козлов А.Ю.
«_____»_____2023 г.

ЗАДАНИЕ

на курсовое проектирование по курсу
«Программирование динамических структур данных»

Студенту Мишихину В.Ю. Группа 21ВП1
Тема проекта Реализация структуры данных «Множество» с использованием контейнерных классов Multiset и queue

Исходные данные (технические требования) на проектирование

Реализовать структуру данных «Множество» и основные операции над множеством (F1-F14) с помощью: односвязного списка, класса «Список», контейнерного класса List, контейнерного класса Set, контейнерного класса Multiset, контейнерного класса queue.

Выполнить сравнение времени выполнения перечисленными выше способами реализации следующих операций: создание множества из N элементов, мощность множества, подмножество $A \subset A$, подмножество $B \subset A$, равенство A=A, равенство B=A, объединение множеств A и B, пересечение множеств A и B, разность A-B, разность B-A, симметричная разность A и B.

Для разработки приложения использовать по выбору любую среду программирования, которая имеет перечисленные в задании контейнерные классы или аналогичные им.

Пользовательский интерфейс должен обеспечивать ввод перечисленных в задании исходных значений, а также удобное представление результирующей информации

Программное обеспечение должно быть полностью отлажено и протестировано, функционировать под управлением ОС Windows.

Объем работы по курсу

1. Расчетная часть

1. Анализ требований к разработке программного обеспечения
2. Проектирование программы
3. Разработка программного обеспечения
4. Тестирование программного обеспечения

2. Графическая часть

1. Диаграмма вариантов использования
2. Диаграмма классов
3. Диаграмма деятельности
4. Диаграмма компонентов

3. Экспериментальная часть

1. Подготовка набора тестовых данных
2. Отладка и тестирование программного обеспечения

Срок выполнения проекта по разделам

1. Анализ требований к разработке ПО	к	20.02.23
2. Проектирование программы	к	01.03.23
3. Разработка программного обеспечения	к	15.04.23
4. Тестирование программного обеспечения	к	15.05.23
5. Оформление пояснительной записки	к	28.05.23
6. Защита курсовой работы	к	29.05.23
	к	

Дата выдачи задания « 08» февраля 2023

Дата защиты проекта « » _____

Руководитель Самуйлов С.В.

Задание получил «08» февраля 2023 г.

Студент Мишкин

2

Реферат

Пояснительная записка содержит 75 листов, 28 рисунков, 11 таблицы, 7 использованных источников, 1 приложение.

МНОЖЕСТВО, ОПЕРАЦИИ НАД МНОЖЕСТВАМИ, ОДНОСВЯЗНЫЕ СПИСКИ, КОНТЕЙНЕРНЫЕ КЛАССЫ, КОНТЕЙНЕРНЫЙ КЛАСС LIST, КОНТЕЙНЕРНЫЙ КЛАСС SET, КОНТЕЙНЕРНЫЙ КЛАСС MULTISET, КОНТЕЙНЕРНЫЙ КЛАСС QUEUE.

Объектом исследования являются основные операции над множеством с использованием различных контейнерных классов.

Целью курсовой работы является разработка приложения для сравнения и анализа времени выполнения операций над множеством с использованием различных контейнерных классов.

Разработка проводилась на языке программирования C++ в среде программирования Visual Studio 2022.

Осуществлено функциональное тестирование разработанного программного обеспечения, которое показало корректность его работы.

Оглавление

Введение

Актуальность темы заключается в том, что в современном информационном обществе обработка данных является неотъемлемой частью многих приложений и систем. Математические множества широко используются в различных областях, таких как анализ данных, машинное обучение, графовые алгоритмы и другие. Операции над математическими множествами, такие как объединение, пересечение, разность и проверка принадлежности, являются важными элементами многих алгоритмов и вычислительных процессов.

Однако, эффективная обработка и выполнение операций над математическими множествами могут быть вызовом, особенно при работе с большими объемами данных. Поэтому важно исследовать и оптимизировать производительность операций над математическими множествами, чтобы обеспечить быструю и эффективную обработку данных.

Основная цель разрабатываемой программы - проанализировать время работы операций над множествами, используя следующие структуры данных: односвязный список, класс «список», контейнерных классов List, Set, Multiset, queue.

Перечень задач, которые необходимо решить для достижения поставленной цели: постановка задачи и анализ предметной области, проектирование, реализация программы, тестирование программы, анализ результатов.

При работе было принято решение использовать такие инструментальные средства, как интегрированная среда разработки Microsoft Visual Studio 2022, инструмент моделирования UML-диаграмм Visual Paradigm.

1 Постановка задачи и анализ предметной области

1.1 Основные понятия и определения

Множество – одно из ключевых понятий, используемых в математике. Обозначает набор, совокупность тех или иных объектов – элементов множества. Два соответствующих компонента равны, если включают в себя одинаковые элементы. Понятие множеств встречается не только в математике, но и в информатике. Этот объект используется в языках программирования.

Множество – своеобразная неупорядоченная совокупность уникальных значений. Элементы в таком «объединении» расположены хаотично. В качестве них выступают:

- строки;
- отдельные символы;
- числовые записи.

Множество – это структура данных, которая представляет собой неорганизованный набор уникальных элементов одного и того же типа. Она имеет тесную связь с математическим понятием теории множеств.

Множества часто используются для хранения объектов пользовательских классов. Для определения множества необходимо указать тип хранимых в нем объектов. К тому же необходимо указать функциональный объект, который будет использоваться для упорядочивания элементов множества.

Множество, которое содержит конечное число элементов, называют конечным. Множество, содержащее бесконечное число элементов, называют бесконечным. Множество, не содержащее ни одного элемента, называется.

Операции над множествами:

- Создание множества
- Мощность множества:

Число элементов множества A обозначается как |A| и называется мощностью

- Подмножество:

Множество A является подмножеством множества B, если всякий элемент множества A является также элементом множества B.

- Равенство множеств:

Множества A и B равны тогда и только тогда, когда A является подмножеством B и B является подмножеством A, то есть элементы множеств A и B совпадают.

- Дополнение:

Для множества A множество B называется дополнением A, если в B включены те и только те элементы, которые не принадлежат A (B=-A, B=$\bar{A}$, B=$\neg$A). Эту операцию еще называют НЕ, то есть говорят B равно НЕ A.

- Объединение множеств:

Объединением множеств A и B называется множество, каждый элемент которого принадлежит либо множеству A, либо множеству B.

- Пересечение множеств:

Пересечением множеств A и B называется множество, каждый элемент которого принадлежит и множеству A, и множеству B.

- Разность множеств:

Разностью множеств A и B называется множество, каждый элемент которого принадлежит множеству A и не принадлежит множеству B.

- Симметричная разность множеств:

Симметричной разностью множеств A и B называется множество, каждый элемент которого принадлежит или множеству A или множеству B, но не обоим множествам одновременно.

Контейнерные классы — это классы, предназначенные для хранения данных, организованных определенным образом. Примерами контейнеров могут служить массивы, линейные списки или стеки. Для каждого типа контейнера определены методы для работы с его элементами, не зависящие от

конкретного типа данных, которые хранятся в контейнере, поэтому один и тот же вид контейнера можно использовать для хранения данных различных типов. Эта возможность реализована с помощью шаблонов классов, поэтому часть библиотеки C++[1], в которую входят контейнерные классы, а также алгоритмы и итераторы, называют стандартной библиотекой шаблонов(STL — Standard Template Library).

Использование контейнеров позволяет значительно повысить надежность программ, их переносимость и универсальность, а также уменьшить сроки их разработки.

STL содержит контейнеры, реализующие основные структуры данных, используемые при написании программ — векторы, двусторонние очереди, списки и их разновидности, словари и множества. Контейнеры можно разделить на два типа:

- Последовательные

  Обеспечивают хранение конечного количества однотипных величин в виде непрерывной последовательности. К ним относятся векторы (vector), двусторонние очереди (deque) и списки (list), а также так называемые адаптеры, то есть варианты, контейнеров — стеки (stack), очереди(queue) и очереди с приоритетами (priority_queue).

- Ассоциативные

  Обеспечивают быстрый доступ к данным по ключу. Эти контейнеры построены на основе сбалансированных деревьев. Существует пять типов ассоциативных контейнеров: словари (map), словари с дубликатами(multlmap), множества (set), множества с дубликатами (multiset) и битовые множества(bitset).

Каждый вид контейнера обеспечивает свой набор действий над данными. Выбор вида контейнера зависит от того, что требуется делать с данными в программе. Контейнерные классы обеспечивают стандартизованный интерфейс при их использовании. Смысл одноименных операций для различных контейнеров одинаков, основные операции

применимы ко всем типам контейнеров. Стандарт определяет только интерфейс контейнеров, поэтому разные реализации могут сильно отличаться по эффективности.

- Контейнерный класс List

Класс list[2] реализован в STL в виде двусвязного списка, каждый узел которого содержит ссылки на последующий и предыдущий элементы. Поэтому операции инкремента и декремента для итераторов списка выполняются за постоянное время, а передвижение на n узлов требует времени, пропорционального n.

После выполнения операций вставки и удаления значения всех итераторов и ссылок остаются действительными.

Список поддерживает конструкторы, операцию присваивания, функцию копирования, операции сравнения и итераторы, аналогичные векторам и очередям.

- Контейнерный класс Set

Множество — это ассоциативный контейнер, содержащий только значения ключей, то есть тип value_type соответствует типу Key. Значения ключей должны быть уникальны. Шаблон множества имеет два параметра: тип ключа и тип функционального объекта, определяющего отношение «меньше»[3].

Интерфейс множества аналогичен интерфейсу словаря. Как и для словаря, элементы в множестве хранятся отсортированными. Повторяющиеся элементы в множество не заносятся.

- Контейнерный класс Multiset

Класс multiset[4] в стандартной библиотеке C++ используется для хранения и извлечения данных из коллекции, в которой значения содержащихся элементов не обязательно должны быть уникальными и в котором они выступают в качестве ключевых значений, в соответствии с которыми данные автоматически упорядочиваются. Ключевое значение элемента в элементе multiset не может быть изменено напрямую. Вместо этого

старые значения необходимо удалить и вставить элементы с новыми значениями.

- Контейнерный класс queue

Очередь — это структура данных (как было сказано выше), которая построена по принципу FIFO (first in — first out: первым пришел — первым вышел).

В очереди, если добавить элемент, который вошел первым, то он выйдет тоже самым первым. Получается, если вы добавите 4 элемента, то первый добавленный элемент выйдет первым

Queue[5] — Класс-шаблон адаптера контейнера, который предоставляет ограничение функциональности для некоторого базового типа контейнера, ограничивая доступ к его передним и задним элементам. Элементы можно добавлять в задней части или удалять с переднего queue края, а элементы можно проверять в любом конце элемента.

1.2 Постановка задачи

Реализовать структуру данных «Множество» и основные операции над множеством (F1-F14) с помощью: односвязного списка[6], класса «Список», контейнерного класса List, контейнерного класса Set, контейнерного класса Multiset, контейнерного класса queue.

Выполнить сравнение времени выполнения перечисленными выше способами реализации следующих операций: создание множества из N элементов, мощность множества, подмножество $A \subset A$, подмножество $B \subset A$, равенство A=A, равенство B=A, объединение множеств A и B, пересечение множеств A и B, разность A-B, разность B-A, симметричная разность A и B.

Для разработки приложения использовать по выбору любую среду программирования, которая имеет перечисленные в задании контейнерные классы или аналогичные им.

Пользовательский интерфейс должен обеспечить ввод перечисленных в задании исходных значений, а также удобное представление результирующей информации.

Программное обеспечение должно быть полностью отлажено и протестировано, функционировать под управлением ОС Windows.

1.3 Анализ требований

1.1.1 Требования к интерфейсу пользователя

Пользовательский интерфейс должен обеспечивать привычное и удобное представление информации, простое и эффективное выполнение основных функций приложения:

- вводить мощность каждого множества
- выводить время выполнения различными методами каждой из перечисленных в задании операции в виде таблицы.

Интерфейс приложения представлен в форме консольного приложения. Вывод временных характеристик осуществляется в таблицу, заголовки столбцы которой – структуры данных, а заголовки строк – наименования основных операций над ними.

1.1.2 Требования к структурам данных

В КР используются следующие СД: односвязный список, класс «Список», контейнерный класс List, контейнерный класс Set, контейнерный класс Multiset, контейнерный класс queue.

- Односвязный список

Объявление:

```
struct Node{
    int val;
    Node *next;
    Node(int _val) : val(_val), next(nullptr) {}
};
Struct list{
    Node* first;
    list() : first(nullptr) {}
}
```

В таблице 1 представлены основные операции данной структуры данных.

Таблица 1 – Операции односвязного списка

Название метода	Назначение
Инициализация списка	Инициализация списка предназначена для создания корневого узла списка, у которого поле указателя на следующий элемент содержит нулевое значение.
Добавление узла в список	Добавление узла включает в себя следующие этапы: создание добавляемого узла и заполнение его поля данных; переустановка указателя узла, предшествующего добавляемому, на добавляемый узел; установка указателя добавляемого узла на следующий узел (тот, на который указывал предшествующий узел).
Удаление узла из списка	В качестве аргументов функции удаления элемента передаются указатель на удаляемый узел, а также указатель на корень списка. Функция возвращает указатель на узел, следующий за удаляемым.
Удаление корня списка	Функция удаления корня списка в качестве аргумента получает указатель на текущий корень списка. Возвращаемым значением будет новый корень списка — тот узел, на который указывает удаляемый корень.
Вывод элементов списка	В качестве аргумента в функцию вывода элементов передается указатель на корень списка. Функция осуществляет последовательный обход всех узлов с выводом их значений. Взаимообмен узлов списка осуществляется путем переустановки указателей. Для этого необходимо определить

	предшествующий и последующий узлы для каждого заменяемого.[5]

- Класс «Список»

Объявление:

```
Struct Node{
    int val;
    Node *next;
    Node *prev;
};
```

В таблице 2 представлены основные операции данной структуры данных.

Таблица 2 – Операции класса «Список»

Название метода	Назначение
Инициализация списка	Инициализация списка предназначена для создания корневого узла списка, у которого поле указателя на следующий элемент содержит нулевое значение.
Добавление узла в список	Метод принимает в качестве аргумента текущий узел со значением по умолчанию, равным NULL. Если узел не передается, добавление нового узла происходит в начало списка.
Удаление узла из списка	В качестве аргументов функции удаления узла передаются указатель на удаляемый узел, а также указатель на корень списка. Функция возвращает указатель на узел, следующий за удаляемым.
Удаление корня	Функция удаления корня ДЛС в качестве аргумента получает указатель на текущий корень списка. Возвращаемым значением будет новый корень списка —

	тот узел, на который указывает удаляемый корень.
Вывод элементов списка	В качестве аргумента в функцию вывода элементов передается указатель на корень списка. Функция осуществляет

Продолжение таблицы 2

Вывод элементов в обратном порядке	Принимает в качестве аргумента указатель на корень списка. Функция перемещает указатель на последний узел списка и осуществляет последовательный обход всех узлов с выводом их значений в обратном порядке.
Взаимообмен узлов	В качестве аргументов функцияпринимает два указателя на обмениваемые узлы, а также указатель на корень списка. Функция возвращает адрес корневого узла списка.Взаимообмен узлов списка осуществляется путем переустановки указателей. Для этого необходимо определить предшествующий и последующий узлы для каждого заменяемого.[6]

- Контейнерный класс List

Объявление:

```
#include <list>
std::list<int>list = { 1, 2, 3, 5 };
```

В таблице 3 представлены основные операции данной структуры данных.

Таблица 3 – Операции контейнерного класса List

Название метода	Назначение

pop_front	Удалить элемент в начале
pop_back	Удалить элемент в конце
erase	Удаление диапазона элементов или одного

Продолжение таблицы 3

push_front	Добавить элемент в начало
push_back	Добавить элемент в конец
front	Обратится к первому элементу
back	Обратиться к последнему элементу
insert	Добавить элемент в какое-то место
copy	Вывести все элементы списка (и не только)
unique	Удалить все дубликаты
merge	Добавление другого списка[7]

- Контейнерный класс Set

Объявление:

#include <set>

set < [тип] ><имя>;

В таблице 4 представлены основные операции данной структуры данных.

Таблица 4 – Операции контейнерного класса Set

Название метода	Назначение
begin	Возвращает итератор, обращающийся к первому элементу в set.
end	Возвращает итератор, который обращается к месту, следующему за последним элементом в контейнере set.
swap	Выполняет обмен элементами между двумя объектами set.
erase	Удаляет элемент или диапазон элементов в наборе с заданных позиций или удаляет

	элементы, соответствующие заданному ключу
find	Возвращает итератор, адресующий расположение элемента в наборе set с ключом, эквивалентным указанному ключу.

Продолжение таблицы 4

count	Возвращает число элементов в контейнере set, ключи которых соответствуют ключу, заданному параметром.[8]

- Контейнерный класс Multiset

Объявление:

#include <set>

multiset< [тип]><имя>;

В таблице 5 представлены основные операции данной структуры данных.

Таблица 5 – Операции контейнерного класса Multiset

Название метода	Назначение
begin	Возвращает итератор, обращающийся к первому элементу в set.
swap	Выполняет обмен элементами между двумя объектами set.
end	Возвращает итератор, который обращается к месту, следующему за последним элементом в контейнере set.
erase	Удаляет элемент или диапазон элементов в наборе с заданных позиций

	или удаляет элементы, соответствующие заданному ключу
find	Возвращает итератор, адресующий расположение элемента в наборе set с ключом, эквивалентным указанному
count	Возвращает число элементов в контейнере set, ключи которых соответствуют ключу, заданному параметром
clear	Стирает все элементы в multiset

- Контейнерный класс queue

Объявление:

#include <queue>

queue <[тип]>name;

В таблице 6 представлены основные операции данной структуры данных.

Таблица 6 – Операции контейнерного класса queue

Название метода	Назначение
back	Возвращает ссылку на последний и наиболее недавно добавленный элемент в конец queue.
empty	Проверяет, является ли queue пустым
front	Возвращает ссылку на первый элемент в начале queue.
size	Возвращает количество элементов в контейнере queue.
pop	Удаляет элемент из начала queue.

push	Добавляет элемент в конец queue.

1.1.3 Требования к программным средствам

Разрабатываемая программа должна выполнять следующие функции:

- создание множества из N элементов
- нахождение мощности множества
- нахождение подмножества A⊂A
- нахождение подмножества B⊂A
- равенство A=A
- равенство B=A
- объединение множеств A и B
- пересечение множеств A и B
- разность A-B
- разность B-A
- симметричная разность A и B
- сравнение времени выполнения различными способами реализации (односвязный список, класс «Список», контейнерный класс List, контейнерный класс Set, контейнерный класс Multiset, контейнерный класс queue) перечисленных выше операций
- позволять вводить мощность множества
- вывод результата

Диаграмма вариантов использования представлена на рисунке 1.

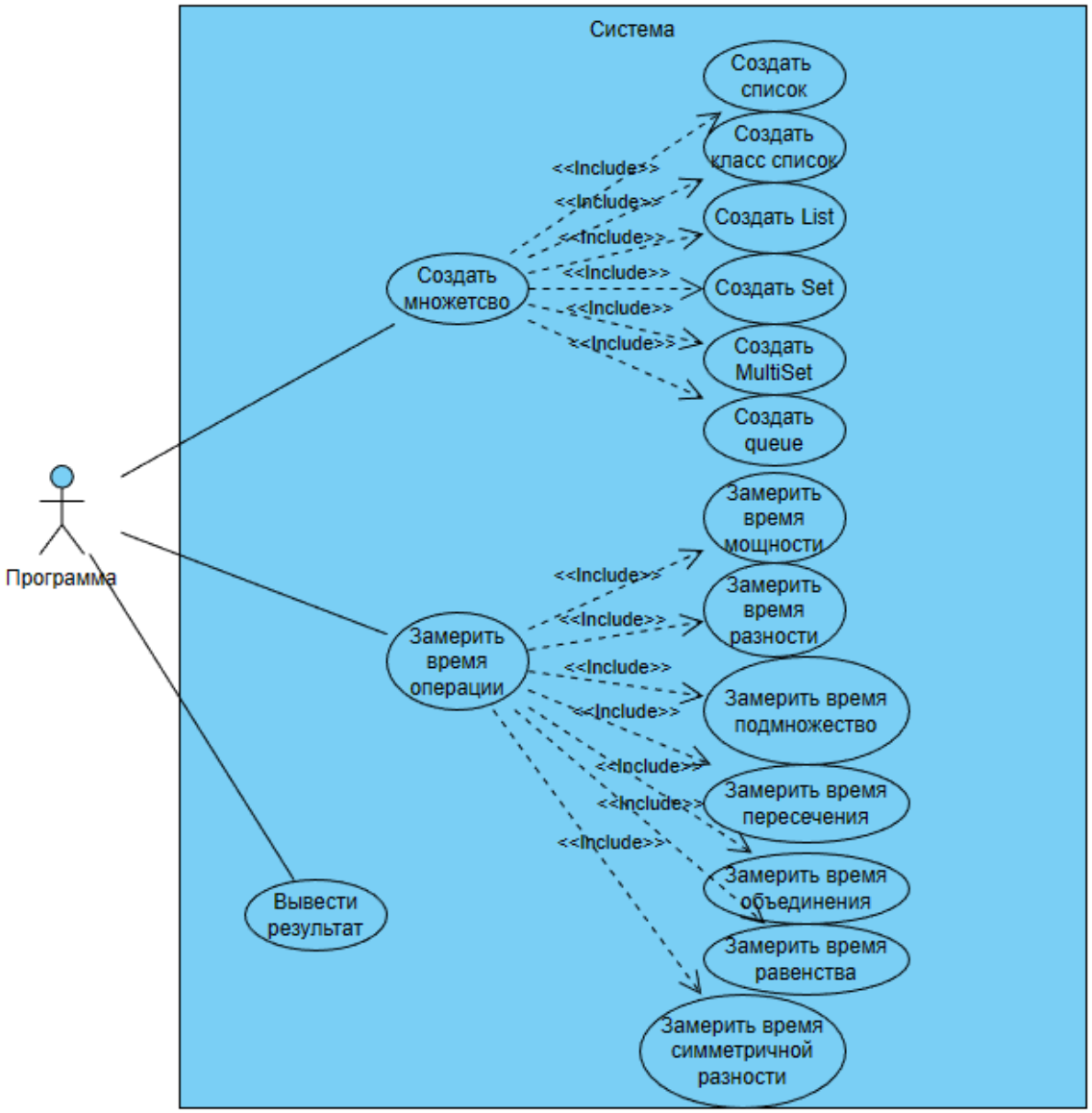

Рисунок 1 – Диаграмма вариантов использования

Описание спецификации прецедентов приведены в таблицах 7–9.

Таблица 7 – Сценарий варианта использования «Найти мощность множества»

Наименование: Найтимощность множества
Краткое описание: система находит мощность множества
Действующие лица: пользователь
Основной поток: 1.Система считает количество элементов в множестве
Постусловие: мощность множества найдена

Таблица 8 – Сценарий варианта использования «Найти разность A-B»

Наименование: Найти разность A-B
Краткое описание: система создает множество, содержащее все элементы, которые принадлежат множеству A и не принадлежат множеству B.
Действующие лица: пользователь
Основной поток: 1.Система создает множество 2.Система добавляет в множество элементы, которые принадлежат множеству A и не принадлежат множеству B.
Постусловие: множество, содержащее разность двух множеств создано

Таблица 9 – Сценарий варианта использования «Найти симметричную разность множеств A и B»

Наименование: Найти симметричную разность множеств A и B
Краткое описание: система создает множество, содержащее все элементы, которые не принадлежат обоим множествам одновременно
Действующие лица: пользователь
Основной поток: 1.Система создает множество всех значений A и B 2.Система создает множество одинаковых значений A и B 2.Системасоздает множество отличных значений между всеми элементами и только одинаковыми элементами
Постусловие: множество, содержащее симметричную разность двух множеств создано

2 Проектирование программы

2.1 Модель интерфейса

В курсовой работе был разработан интерфейс, представленный на рисунке 2.

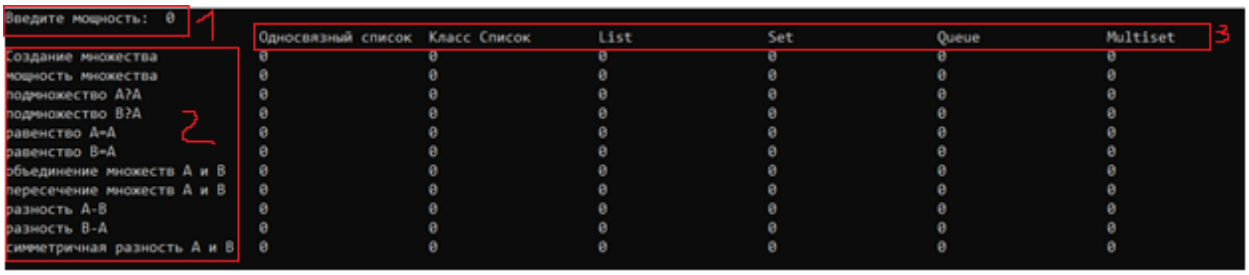

Рисунок 2 – Модель интерфейса

Интерфейс условно поделен на четыре части: поле, вводимое пользователем (1), которое определяет мощность создаваемых множеств и таблица, выводящая результаты работы программы. Таблица состоит из названий структур данных (2), использующихся в программе, операций, производимых над ними (3) и временем выполнения этих операций для каждой структуры (невыделенная область).

2.2 Проектирование структур данных

В КР используются следующие СД: односвязный список, класс «Список», контейнерный класс List, контейнерный класс Set, контейнерный класс Multiset, контейнерный класс queue.

Ниже приведено описание СД – «Односвязный список» и функции «Создание множества».

Листинг 1. Код описания СД - «Односвязный список»

```
Node
{
    Int val;
    Elem* next;
};
```

Пояснительный текст листинга 1:

val–элемент множества;

Node* next – указатель на следующий элемент.

Листинг 2. Код описания функции «Создание множества»:

```
Node* create(int n, int min, int max)
{
	Node* start = NULL;
	Node* prev;
	Int a;
	Int i = 0;
	while(i< n)
	{
		prev = start;
		a = random(min, max);
		start = push_up(start, a);
		if (start != prev) i++;
	}
	Return start;
}
```

Пояснительный текст листинга 2:

Node* start – указатель на начало;

a – элемент массива;

i – вспомогательная переменная для прохода по множеству;

push_up – функция, добавляющая элемент в начало множества;

random(min, max) – функция, генерирующая рандомное число в диапозоне[min;max].

Ниже приведено описание СД – Класс «Список» и функции «Вывод в строку».

Листинг 3. Код описания СД - Класс «Список»

```
struct Node {
	int val;
	Node* next;
	Node(int _val) : val(_val), next(nullptr) {}
};

class listok {
public:
	Node* first;
	listok() {
		this->first = NULL;
	}
	listok(Node* head) {
		this->first = head;
	}
```

Пояснительный текст листинга 3:

val–элемент множества;

Node* next – указатель на следующий элемент;

Node* first – указатель на голову списка.

Листинг 4. Код описания функции «Мощность множества»:

```
string listok::printstr(Node* headptr, string c) {
   string res;

   if (empty(headptr)) {
      return " ";
   }
   while (headptr) {
      int xx = headptr->val;
```

```
        res += to_string(xx) + c;
        headptr = headptr->next;
    }
    res.pop_back();
    return res;
};
```

Пояснительный текст листинга 4:

res – строка элементов;

pop_back()–функция удаления последнего символа в строке.

Ниже приведено описание СД – контейнерный класс List и функции «Подмножество».

Листинг 5. Код описания СД - Контейнерный класс List

```
class PlentyList {
public:
PlentyList() {
        std::list<int> lst;
}

}
```

Пояснительный текст листинга 5:

PlentyList–множество;

lst– объект класса list.

Листинг 6. Код описания функции «Подмножество»:

```
bool PlentyList::is_subset(std::list<int> lsta, std::list<int> lstb) {
        if (lsta.size() > lstb.size()) {
                return false;
        }
        std::list<int>::iterator ita;
        std::list<int>::iterator itb;

        for (ita = lsta.begin(); ita != lsta.end(); ++ita) {
```

```
            if (is_unique(lstb, *ita)) {
                return false;
            }
        }
        return true;

    }
```

Пояснительный текст листинга 6:

Lsta,lstb – Множества А и Б;

std::list<int>::iterator ita (itb); – итераторы;

Is_unique – функция, проверяющая уникальность элемента.

Ниже приведено описание СД – контейнерный класс Set и функции «Объединение».

Листинг 7. Код описания СД - Контейнерный класс Set

```
class Plenty {
public:
    Plenty() {
        set<int> lst;
    }
}
```

Пояснительныйтекстлистинга7:

Plenty–множество;

lst – объект класса set.

Листинг 8. Код описания функции «Объединение»:

```
Plenty Plenty::combine(set<int> lsta, set<int> lstb) {
    Plenty plenty;
    set<int>::iterator it;
    for (it = lsta.begin(); it != lsta.end(); ++it) {
        plenty.push_up(*it);
    }
    for (it = lstb.begin(); it != lstb.end(); ++it) {
        plenty.push_up(*it);
    }
    return plenty;

 }
```

Пояснительный текст листинга 8:

Push_up – функция, добавляющая элемент;

Plenty plenty – множество, содержащее объединение двух других.

Ниже приведено описание СД – контейнерный класс Multiset и функции «Мощность».

Листинг 9. Код описания СД - Контейнерный класс Multiset

```
class Multiset {
public:
   multiset<int> s;
}
```

Пояснительный текст листинга 9:

plenty–множество.

Листинг 10. Код описания функции «Мощность»:

```
int Multiset::size() {
      return s.size();

}
```

Пояснительный текст листинга 10:

size – функция коллекции, возвращает длину объекта Multiset;

Ниже приведено описание СД – контейнерный класс queue и функции «Разность».

Листинг 11. Код описания СД - Контейнерный класс

```
class QueSet {
public:
   queue<int> elements;
}
```

Пояснительный текст листинга11:

QueSet–множество;

elements- объект коллекции queue.

Листинг12. Код описания функции «Разность»:

```
QueSet QueSet::difference(QueSet s) {
        QueSet result;
        queue<int> temp = elements;
        while (!temp.empty()) {
            int x = temp.front();
            if (!s.contains(x)) {
                result.add(x);
            }
            temp.pop();
        }
        return result;
    }
```

Пояснительный текст листинга 12:

result – множество результата разности;

pop – функция, удаления элемента;

contains – функция, проверяющая наличие элемента.

2.3 Структура программного обеспечения

Структура программного обеспечения описана диаграммой классов, представленной на рисунке 3.

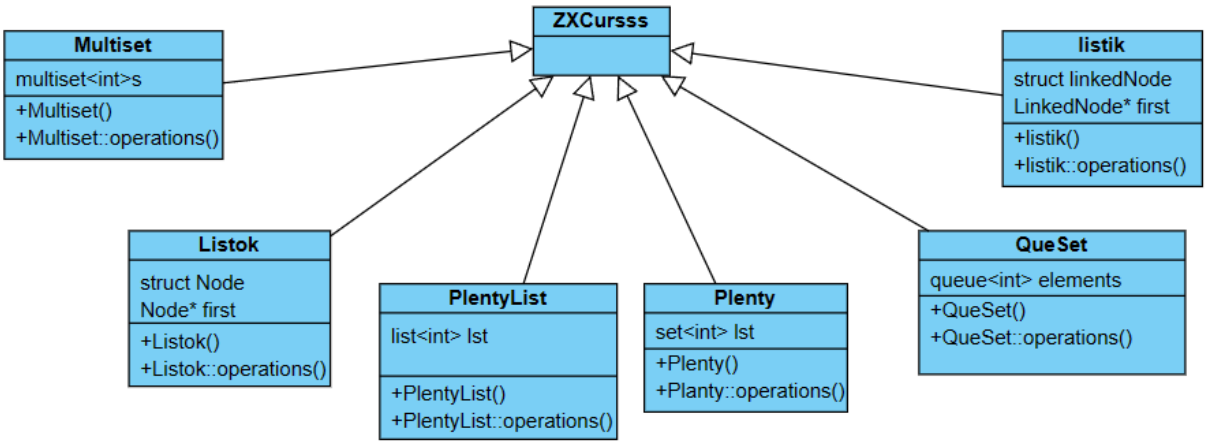

Рисунок 3 – Диаграмма классов

Ниже представлено краткое описание спецификаций каждого класса

Описание классов:

1. Класс ZXCursss – представляет собой главную программу

2. Класс listik – представляет собой односвязный список

Члены класса:

int val – значение

LinkedNode* next – указатель на следующий узел

LinkedNode* first – указатель на первый узел

Методы:

LinkedNode* create(int x, int min, int max); – генерирование рандомного множества

listik() : first(nullptr), last(nullptr) {} – создание пустого множества

bool empty(LinkedNode* headptr); – проверка пустого множества

bool not_belong_to(LinkedNode* p, int _val); – проверка принадлежит ли элемент множеству

bool is_subset(LinkedNode* heada, LinkedNode* headb); - проверка является ли одно множество подмножеством другого.

bool is_equal(LinkedNode* heada, LinkedNode* headb);- проверка равны ли множества

int count(LinkedNode* headptr);- мощность множества

string printstr(LinkedNode* headptr, string c); - вывод множества

LinkedNode* deletelist(LinkedNode*& headptr); - удаление множества

LinkedNode* add_to_begin(LinkedNode*& p, int _val);- добавление элемента в начало множества

LinkedNode* combine(LinkedNode* heada, LinkedNode* headb);- объединение множеств

LinkedNode* intersection(LinkedNode* heada, LinkedNode* headb);- пересечение множеств

LinkedNode* difference(LinkedNode* heada, LinkedNode* headb);- разность множеств

LinkedNode* simmetric(LinkedNode* heada, LinkedNode* headb);– симметричная разность

3. Класс listok– представляет собой класс на основе связного списка.

Члены класса:

Node* first;- указатель на голову

int val; - значение

Node* next; - указатель на следующий элемент

Методы:

listok(Node* head) - конструктор

bool empty(Node* headptr);- проверка на пустоту

bool not_belong_to(Node* p, int _val); - проверка на принадлежность элемента множеству

bool is_subset(Node* heada, Node* headb);- проверка является ли одно множество подмножеством другого.

bool is_equal(Node* heada, Node* headb); проверка равны ли множества

int count(Node* headptr); - мощность множества

string printstr(Node* headptr, string c); - вывод множество строкой

Node* create(int x, int min, int max, int param); - создание случайного множества

Node* add_to_begin(Node*& p, int _val);- добавить элемент в начало

Node* combine(Node* heada, Node* headb); - объединение множеств

Node* intersection(Node* heada, Node* headb);- пересечение множеств

listok* difference(Node* heada, Node* headb); - разность множеств

listok* simmetric(Node* heada, Node* headb);- симметричная разность

~listok()- деструктор

4. Класс PlentyList– представляет собой класс на основе List

Члены класса:

std::list <int> lst;– множество

Методы:

PlentyList() – конструктор(создание пустого множества)

bool is_empty(std::list<int> lst);- проверка на пустоту

void push_up(int val);- добавить элемент в начало

std::string print(std::list<int> lst, char c); - вывод множества в строку

bool is_unique(std::list<int> lst, int val);- проверка на уникальность элемента

int count(); - мощность множества

void create(int x, int min, int max, int last); - создание случайного множетсва

bool is_subset(std::list<int> lsta, std::list<int> lstb);- является ли одно множество подмножеством другого

bool is_equal(std::list<int> lsta, std::list<int> lstb);- равны ли множества

PlentyList combine(std::list<int> lsta, std::list<int> lstb);- объединение множеств

PlentyList intersection(std::list<int> lsta, std::list<int> lstb);- пересечение множеств

PlentyList difference(std::list<int> lsta, std::list<int> lstb);- разность множеств

PlentyList simmetric(std::list<int> lsta, std::list<int> lstb); симметричная разность

5. Класс Plenty– представляет собой контейнер на основе Set

Члены класса:

set<int>lst– множество

Методы:

bool is_empty(std:: set <int> lst);- проверка на пустоту

void push_up(int val);- добавить элемент в начало

std::string print(std:: set <int> lst, char c); - вывод множества в строку

bool is_unique(std:: set <int> lst, int val);- проверка на уникальность элемента

int count(); - мощность множества

void create(int x, int min, int max, int last); - создание случайного множетсва

bool is_subset(std:: set <int> lsta, std:: set <int> lstb);- является ли одно множество подмножеством другого

bool is_equal(std:: set<int> lsta, std:: set <int> lstb);- равны ли множества

Plenty combine(std:: set <int> lsta, std:: set <int> lstb);- объединение множеств

Plenty intersection(std:: set <int> lsta, std:: set <int> lstb);- пересечение множеств

Plenty difference(std::set<int> lsta, std:: set <int> lstb);- разность множеств

Plenty simmetric(std:: set <int> lsta, std:: set <int> lstb); симметричная разность

6. Класс Multiset– представляет собой контейнер на основе multiset

Члены класса:

multiset<int>plenty– множество

Методы:

bool is_empty(std:: multiset <int> lst);- проверка на пустоту

void push_up(int val);- добавить элемент в начало

std::string print(std:: multiset <int> lst, char c); - вывод множества в строку

bool is_unique(std:: multiset <int> lst, int val);- проверка на уникальность элемента

int count(); - мощность множества

void create(int x, int min, int max); - создание случайного множетсва

bool is_subset(std:: multiset <int> lsta, std:: multiset <int> lstb);- является ли одно множество подмножеством другого

bool is_equal(std:: multiset <int> lsta, std:: multiset <int> lstb);- равны ли множества

Multiset combine(std:: multiset <int> lsta, std:: multiset <int> lstb);- объединение множеств

Multiset intersection(std:: multiset <int> lsta, std:: multiset <int> lstb);- пересечение множеств

Multiset difference(std:: multiset <int> lsta, std:: multiset <int> lstb);- разность множеств

Multiset simmetric(std:: multiset <int> lsta, std:: multiset <int> lstb); симметричная разность

7. Класс QueSet– представляет собой очередь

Члены класса:

queue <int>plenty– множество

Методы:

bool is_empty(std:: queue <int> lst);- проверка на пустоту

void push_up(int val);- добавить элемент в начало

std::string print(std:: queue <int> lst, char c); - вывод множества в строку

bool is_unique(std:: queue <int> lst, int val);- проверка на уникальность элемента

int count(); - мощность множества

void create(int x, int min, int max, int last); - создание случайного множетсва

bool is_subset(std:: queue <int> lsta, std:: queue <int> lstb);- является ли одно множество подмножеством другого

bool is_equal(std:: queue <int> lsta, std:: queue <int> lstb);- равны ли множества

QueSet combine(std:: queue <int> lsta, std:: queue <int> lstb);- объединение множеств

QueSet intersection(std:: queue <int> lsta, std:: queue <int> lstb);- пересечение множеств

QueSet difference(std:: queue <int> lsta, std:: queue <int> lstb);- разность множеств

QueSet simmetric(std:: queue <int> lsta, std:: queue <int> lstb); симметричная разность

3 Реализация программы

3.1 Кодирование

В ходе выполнения курсовой работы было разработано приложение «Реализация структуры данных «Множество» с использованием контейнерных классов Multiset и queue», код которого приведен в приложении А.

Для алгоритма операции создать множество была разработана диаграмма деятельности (рисунок 4).

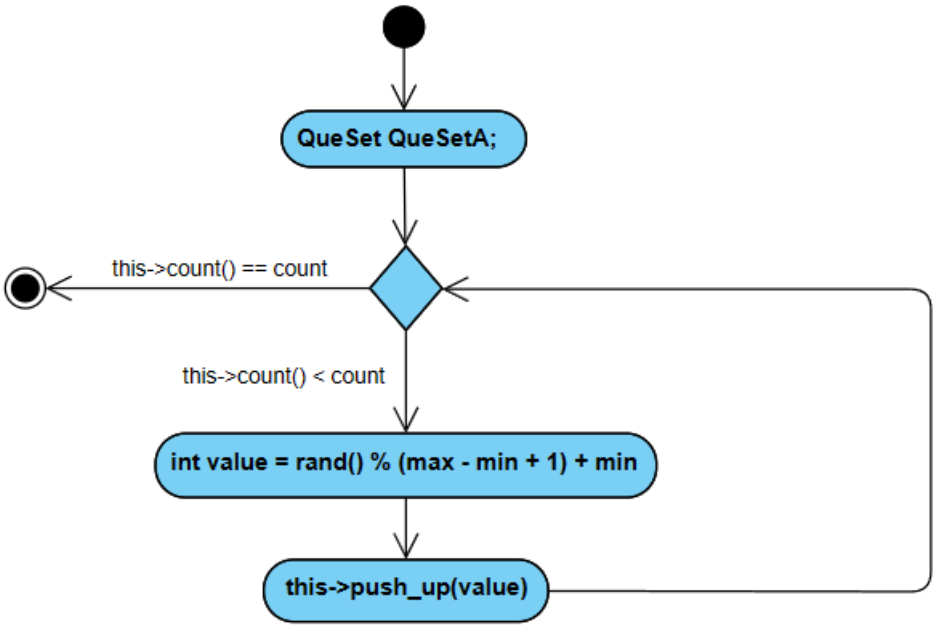

Рисунок 4 – Диаграмма деятельности операции «Создать множество»

На диаграмме используются следующие переменные и методы:

- value – случайно генерируемое значение
- count – передающийся в операцию параметр, кол-во элементов, которые должны быть добавлены в множество
- count() – функция, возвращающая длину множество в момент вызова
- push_up(value) – функция добавления элемента в множество

3.2 Диаграмма компонентов

В процессе выполнения курсовой работы была составлена диаграмма компонентов, которая отображает разбиение программной системы на структурные компоненты и связи между компонентами (рисунок 5).

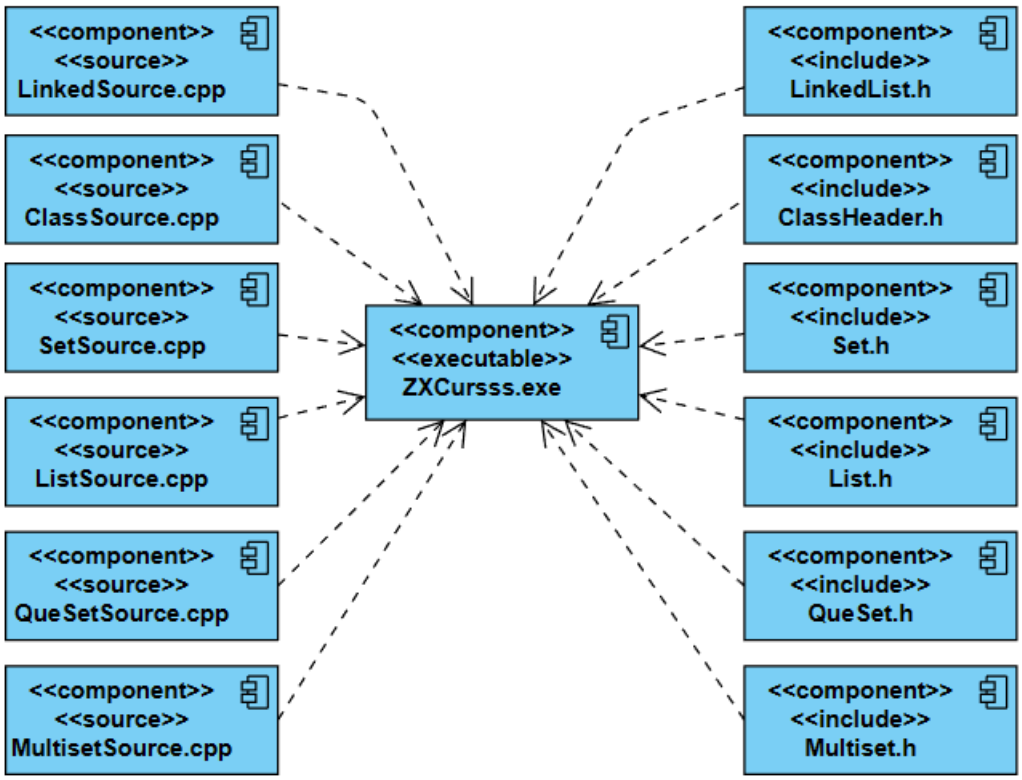

Рисунок 5 – Диаграмма компонентов

Описание компонентов приведено в таблице 10.

Таблица 10 – Компоненты диаграммы

Компоненты	Назначение
Linkedlist.h	Заголовочный файл класса listik
LinkedSource.cpp	Исходный файл класса listik
ClassHeader.h	Заголовочный файл класса listok
ClassSource.cpp	Исходный файл класса listok
List.h	Заголовочный файл класса PlentyList
ListSource.cpp	Файл реализации класса PlentyList
Set.h	Заголовочный файл класса Plenty

Продолжение таблицы 10

SetSource.cpp	Файл реализации класса Plenty
Multiset.h	Заголовочный файл класса Multiset
MultisetSource.cpp	Файл реализации класса Multiset
QueSet.h	Заголовочный файл класса QueSet
QueSetSource.cpp	Файл реализации класса QueSet
ZXCursss.cpp	Главная программа
ZXCursss.exe	Исполняемый файл приложения

4 Тестирование

4.1 Функциональное тестирование

Функциональное тестирование — это процесс обеспечения качества в рамках цикла разработки программного обеспечения, необходимый для проверки реализуемости функциональных требований, согласно спецификации тестируемого программного обеспечения. Функциональное тестирование проводится для оценки соответствия системы или компонента заданным функциональным требованиям.

Функциональное тестирование является тестированием черного ящика, в связи с чем функциональность программного обеспечения можно протестировать, не зная его внутренней работы. Это снижает требования к тестировщикам в части знания языков программирования или конкретных аспектов реализации программного обеспечения.

4.2 Тестирование приложения

В курсовой работе было выполнено функциональное тестирование разработанного программного обеспечения. Результаты тестирования приведены в таблице 11.

Таблица 11 – Функциональное тестирование

Состав теста	Ожидаемый результат	Наблюдаемый результат
Введенный размер множества: 1000	Размер множества подойдет условию, программа выдаст результат	Размер множества подошел условию, программа выдала результат (рисунок 6)
Введенный размер множества: 400	Размер множества не подойдет условию, программа не выдаст результат	Размер множества не прошел условие, программа выдала предупреждение (рисунок 7)

Продолжение таблицы 11

Введенный размер множества: 1500	Размер множества подойдет условию, программа выдаст результат	Размер множества подошел условию, программа выдала результат (рисунок 8)
Введенный размер множества: 1999	Размер множества подойдет условию, программа выдаст результат	Размер множества подошел условию, программа выдала результат (рисунок 9)
Введенный размер множества: -500	Размер множества не подойдет условию, программа не выдаст результат	Размер множества не прошел условие, программа выдала предупреждение (рисунок 10)
Введенный размер множества: 999	Размер множества подойдет условию, программа выдаст результат	Размер множества подошел условию, программа выдала результат (рисунок 11)
Введенный размер множества: 2000	Размер множества подойдет условию, программа выдаст результат	Размер множества подошел условию, программа выдала результат (рисунок 12)
Введенный размер множества: 9999	Размер множества не подойдет условию, программа не выдаст результат	Размер множества не прошел условие, программа выдала предупреждение (рисунок 13)
Введенный размер множества: 2999	Размер множества подойдет условию, программа выдаст результат	Размер множества подошел условию, программа выдала результат(рисунок 14)
Введенный размер множества: 0	Размер множества не подойдет условию, программа выдаст предупреждение	Размер множества не прошел условие, программа выдала предупреждение(рисунок 15)
Введенный размер множества: 200	Размер множества не подойдет условию, программа не выдаст результат	Размер множества не прошел условие, программа выдала предупреждение(рисунок 16)
Введенный размер множества: 15000	Размер множества не подойдет условию, программа не выдаст результат	Размер множества не прошел условие, программа выдала

		предупреждение (рисунок 17)

Ниже, на рисунках 6 - 17, приведены скриншоты, отражающие результаты работы программы в процессе тестирования.

```
1000
Название     LinkedList   ClassList    List         Set          Multiset     QueSet
Создание       1.861300     3.695600   721.842700   285.833100   221.882100   424.169400
Мощность       0.024100     0.004000     0.000300     0.000200     0.000200     0.000200
Подм-во A и A  1.217700     1.442800   424.387500   496.779300   425.060500   735.786900
Подм-во B и A  1.204400     0.028300     1.035000     1.593100     1.543700   741.559400
A=A            1.227500     1.446600   391.709900   473.507000   410.172000   725.804400
A=B            1.195100     0.016200     1.719200     2.432400     2.240300   747.568900
Об-ие A и B    2.577000     6.244400   879.578800     3.344300   804.805600  1128.916800
Пер-ние A и B  2.999800     3.409700   430.798800   532.328700   419.054800  1133.526900
Раз-ть A-B     1.204600     5.300303   591.473600   542.661300   588.908200   735.466700
Раз-ть B-A     1.259500     4.381300   586.855100   548.527300   579.874300   734.312900
Симмет         7.198200    11.711800  2032.698300   670.128400  2000.014300  3012.284500
```

Рисунок 6 – Результат работы 1 теста

```
400
Не корректный ввод множетсва, введите значение от 500 и до 3000
```

Рисунок 7 – Результат работы 2 теста

```
1500
Название     LinkedList   ClassList    List         Set          Multiset     QueSet
Создание       3.825600     7.805900  1346.602000   678.336400   514.744600   975.322300
Мощность       0.011600     0.005900     0.000300     0.000300     0.000200     0.000200
Подм-во A и A  3.260700     3.607300  1050.500100  1097.787700   943.810000  1673.377400
Подм-во B и A  3.517800     0.029400     2.216400     2.302900     2.272000     3.115800
A=A            3.117900     3.635900  1058.676300  1075.881900   951.374800  1677.633200
A=B            3.485800     0.042500     3.692000     3.628200     3.678100     4.125700
Об-ие A и B    7.571600    15.058000  2097.225100     5.110500  1828.293900  3490.354400
Пер-ние A и B  8.280200     7.477800   944.275000  1258.990800   965.638400  1788.712700
Раз-ть A-B     3.716300    11.077600  1318.977600  1233.886500  1323.156600  2518.151000
Раз-ть B-A     3.617900    10.878700  1291.116000  1225.801100  1338.472300  2550.191900
Симмет        19.010600    28.816600  4496.347600  1525.545100  4564.525300  8552.312200
```

Рисунок 8 – Результат работы 3 теста

```
1999
Название     LinkedList   ClassList    List         Set          Multiset     QueSet
Создание       5.743400    13.601200  2624.300500  1170.922800   885.853800  1689.653800
Мощность       0.007900     0.007700     0.000400     0.000400     0.000100     0.000300
Подм-во A и A  6.151000    14.080700  1891.424000  1952.569600  1634.646600  2935.202800
Подм-во B и A  6.240100     0.328900     2.612500     3.168900     2.856500     4.247700
A=A            6.194000     9.646400  1421.855300  1907.075600  1634.600200  2912.719400
A=B            6.168800     0.102100     3.646500     5.164100     5.106400     5.472700
Об-ие A и B   13.202900    26.346000  3220.926100     7.129200  3215.777600  6008.471900
Пер-ние A и B 13.671800    12.354700  1654.409700  2165.214100  1637.250600  3092.117800
Раз-ть A-B     6.307600    18.546600  2298.528600  2186.460300  2354.460700  4346.700800
Раз-ть B-A     6.142900    18.186700  2308.811400  2219.347500  2316.397000  4328.895000
Симмет        33.025100    51.897900  7836.187900  2716.741600  7894.031800 14756.441300
```

Рисунок 9 – Результат работы 4 теста

```
-500
Не корректный ввод множетсва, введите значение от 500 и до 3000
```

Рисунок 10 – Результат работы 5 теста

```
999
Название     LinkedList   ClassList    List         Set          Multiset     QueSet
Создание          1.432200     2.643300   581.750600   397.629700   229.155100   429.066000
Мощность          0.004100     0.003000     0.000600     0.000200     0.000100     0.000200
Подм-во A и A     1.205000     1.227000   486.983200   508.646900   417.922500   733.114000
Подм-во B и A     1.200500     0.008200     1.431000     1.520900     1.412500   749.709600
A=A               1.209900     1.229300   467.134400   486.535000   413.048200   746.332000
A=B               1.206500     0.013500     2.257800     2.439600     2.196900   735.317700
Об-ие A и B       2.666800     5.867200  1215.752400     3.293800   826.390000  1136.872100
Пер-ние A и B     2.816100     3.103300   567.706900   546.832300   426.629400  1118.858700
Раз-ть A-B        1.196800     4.432200   782.533200   549.882400   572.574700   773.712800
Раз-ть B-A        1.208700     4.765200   770.664100   535.135200   582.759800   748.917900
Симмет            6.640500    11.698400  2666.699200   676.358700  1971.105900  3056.742700
```

Рисунок 11– Результат работы 6 теста

```
2000
Название     LinkedList   ClassList    List         Set          Multiset     QueSet
Создание          5.817500    13.642300  2573.328700  1241.537000   888.275500  1710.371200
Мощность          0.007700     0.008100     0.000400     0.000100     0.000200     0.000500
Подм-во A и A     5.237500     6.159800  1853.633500  2017.024500  1677.267500  2950.105700
Подм-во B и A     5.202000     0.029000     3.091400     3.017000     2.837000     4.158400
A=A               5.189900     6.202400  1498.474100  2048.322900  1688.896700  2965.693000
A=B               5.299500     0.042200     3.731600     5.365900     6.043500     5.418700
Об-ие A и B      10.641700    44.474700  3389.213300     7.079000  3286.864100  6117.001000
Пер-ние A и B    13.675700    19.479900  1759.162400  2298.262900  1682.420800  3108.782700
Раз-ть A-B        6.372300    18.752600  2424.716900  2390.185900  2366.928300  4403.313300
Раз-ть B-A        6.257900    17.912400  2413.527100  2390.563600  2373.055800  4402.671900
Симмет           33.255400    54.730500  8239.572100  2810.183700  8088.329900 15148.754400
```

Рисунок 12 – Результат работы 7 теста

```
9999
Не корректный ввод множетсва, введите значение от 500 и до 3000
```

Рисунок 13 – Результат работы 8 теста

```
2999
Название     LinkedList   ClassList    List         Set          Multiset     QueSet
Создание         13.368800    31.881000  4734.556000  2685.685400  2045.456200  3846.909900
Мощность          0.014300     0.015600     0.000400     0.000100     0.000100     0.000200
Подм-во A и A    11.438200    17.166200  3359.861600  4408.062000  3765.066400  6715.722400
Подм-во B и A    14.356200     0.191200     3.825900     4.852200     4.193500     7.025100
A=A              14.787900    17.041900  3309.492400  4425.599000  3770.410000  6742.135800
A=B              14.940200     0.353200     6.095900     8.869000     6.938700     8.200000
Об-ие A и B      31.459100    64.817000  7572.365800    12.550100  7327.256200 14168.897800
Пер-ние A и B    39.811300    31.451300  3872.589400  4954.818000  3854.887600  7340.437600
Раз-ть A-B       20.806200    48.789800  5380.120000  4974.259300  5533.827800 10499.106000
Раз-ть B-A       15.529500    56.535500  5434.250900  4943.063700  5420.945500 10478.245400
Симмет           84.301600   141.794300 18478.281200  6009.826000 17942.698600 34761.294100
```

Рисунок 14 – Результат работы 9 теста

```
0
Не корректный ввод множетсва, введите значение от 500 и до 3000
```

Рисунок 15 – Результат работы 10 теста

```
200
Не корректный ввод множетсва, введите значение от 500 и до 3000
```

Рисунок 16 – Результат работы 11 теста

```
15000
Не корректный ввод множетсва, введите значение от 500 и до 3000
```

Рисунок 17 – Результат работы 12 теста

В ходе выполнения тестирования несовпадения ожидаемого и наблюдаемого результата не выявлены. Следовательно, можно сделать вывод, что программа работает корректно.

5 Анализ результатов

В задании к данной работе требовалось сравнить время выполнения операций над множеством с использованием различных структур данных.

Ниже на рисунках 18-28 будут приведены итоги сравнений в табличном и графическом виде.

Название	LinkedList	ClassList	List	Set	Multiset	QueSet
Создание	0,3727	0,5834	145,6438	95,2076	71,2698	139,8959

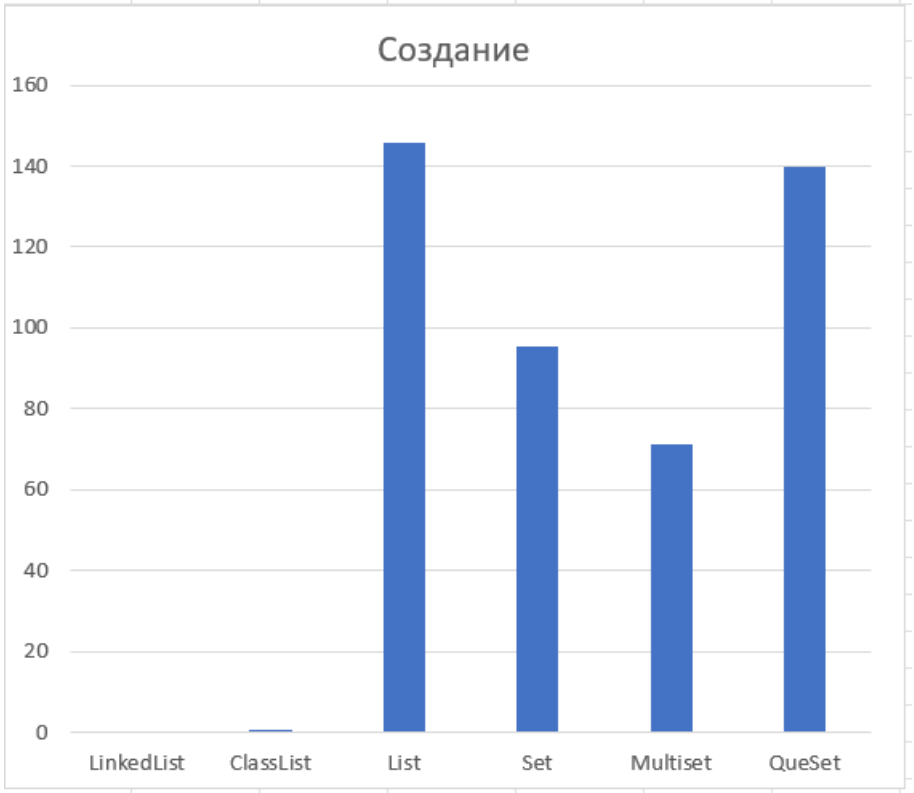

Рисунок 18 - Результат работы операции “Создание”

Операция “Создание” демонстрирует, что структуры данных использующие односвязный список намного быстрее генерируют множество нежели контейнерные классы.

Название	LinkedList	ClassList	List	Set	Multiset	QueSet
Мощность	0,0023	0,0015	0,0004	0,0004	0,0001	0,0002

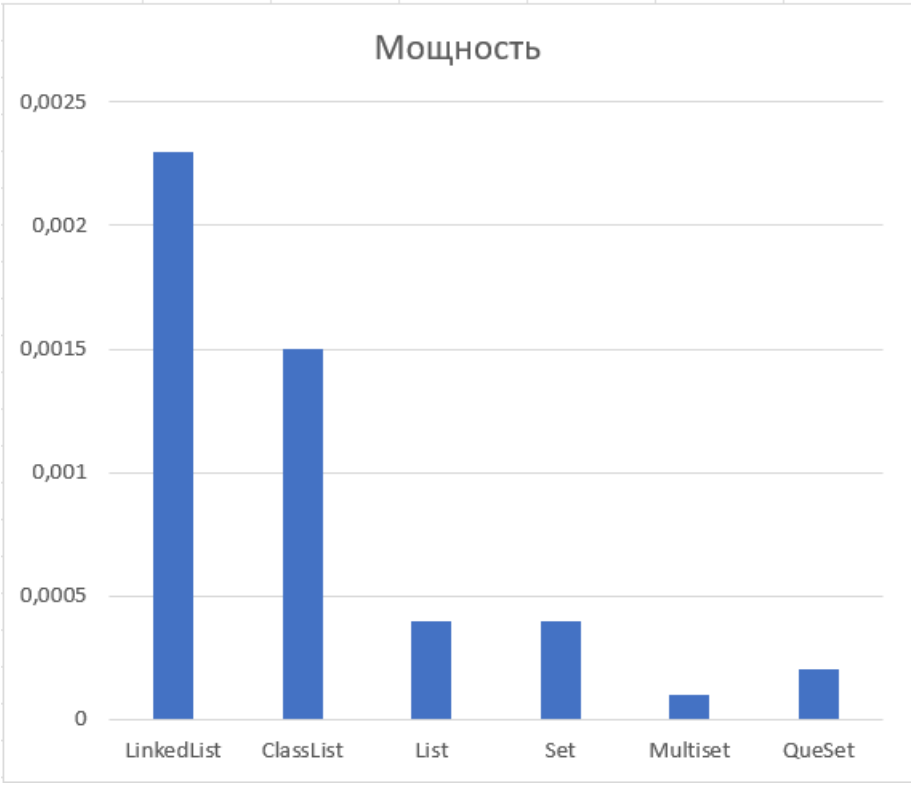

Рисунок 19 - Результат работы операции “Мощность”

Операция “Мощность” во всех случаях выполняется очень быстро, так как требует наименьшее количество проходов по множеству.

Название	LinkedList	ClassList	List	Set	Multiset	QueSet
Подм-во А - А	0,2234	0,2182	140,6538	196,6312	140,9916	266,1127

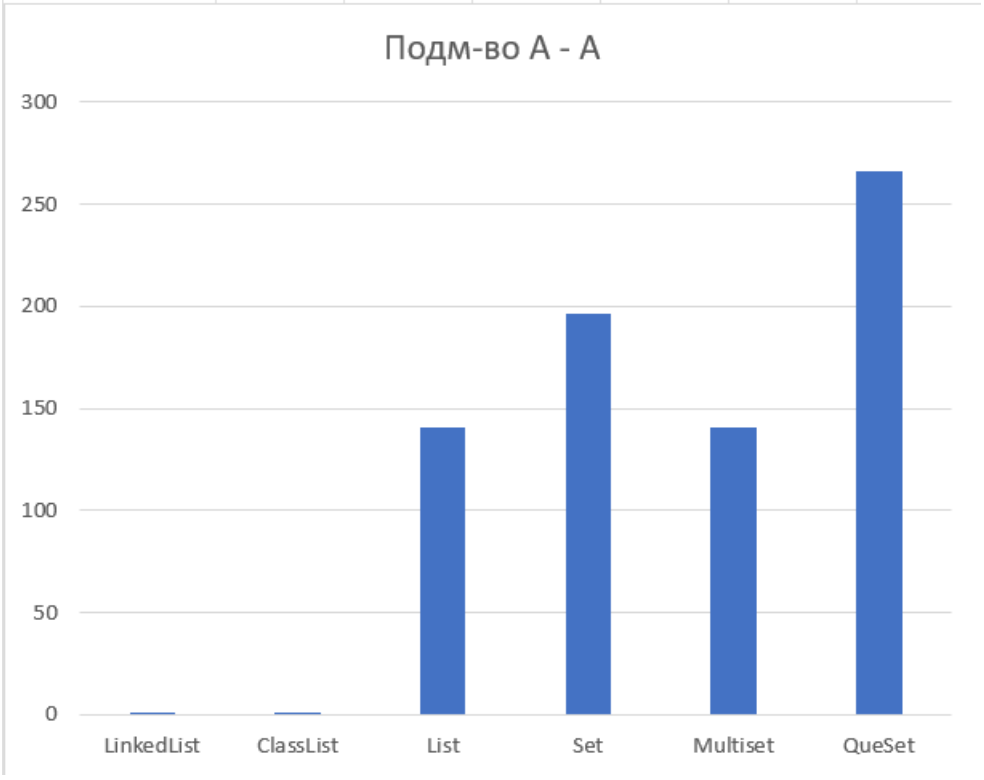

Рисунок 20 - Результат работы операции “Подмножество А-А”

В операции проверки на подмножество результаты демонстрируют превосходство односвязного списка и класса на его основе перед другими структурами данных.

Название	LinkedList	ClassList	List	Set	Multiset	QueSet
Подм-во A-B	0,2275	0,0043	1,0702	0,9661	0,8818	1,4298

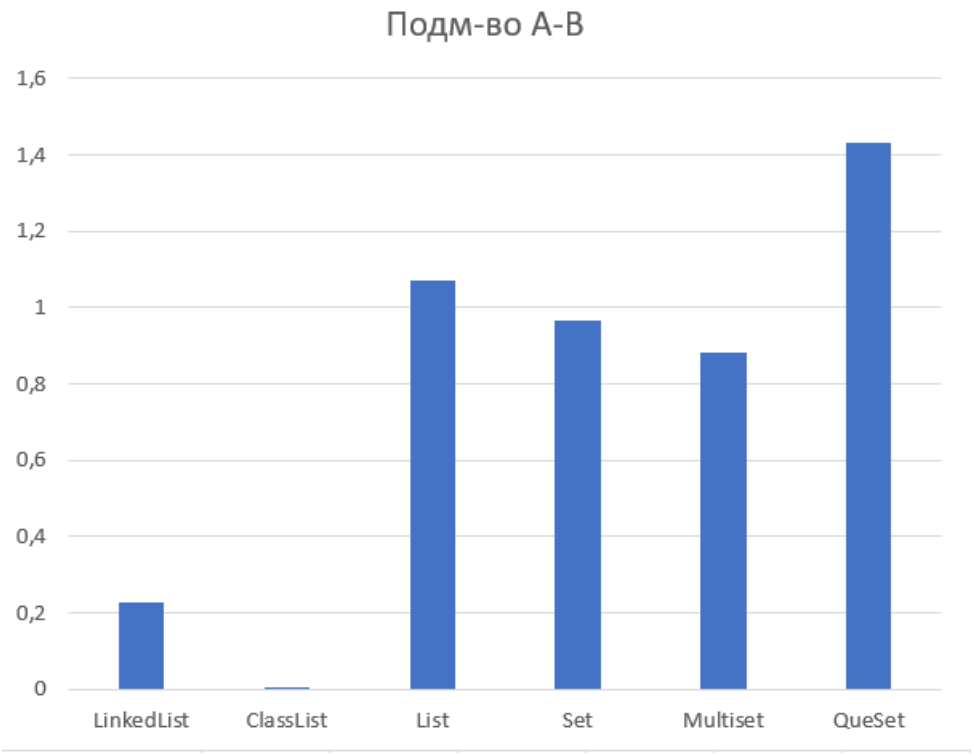

Рисунок 21 - Результат работы операции “Подмножество A-B”

Проверка на подмножество двух случайных множеств происходит в большинстве случаев довольно быстро, так как мы прекращаем итерации как только встречаем в множестве A первый элемент, которого нет в B. Но не смотря на это результаты все равно соответствуют ожиданиям и схожи с результатами других операций.

Название	LinkedList	ClassList	List	Set	Multiset	QueSet
A=A	0,2256	0,2215	166,817	185,1398	137,3767	259,1604

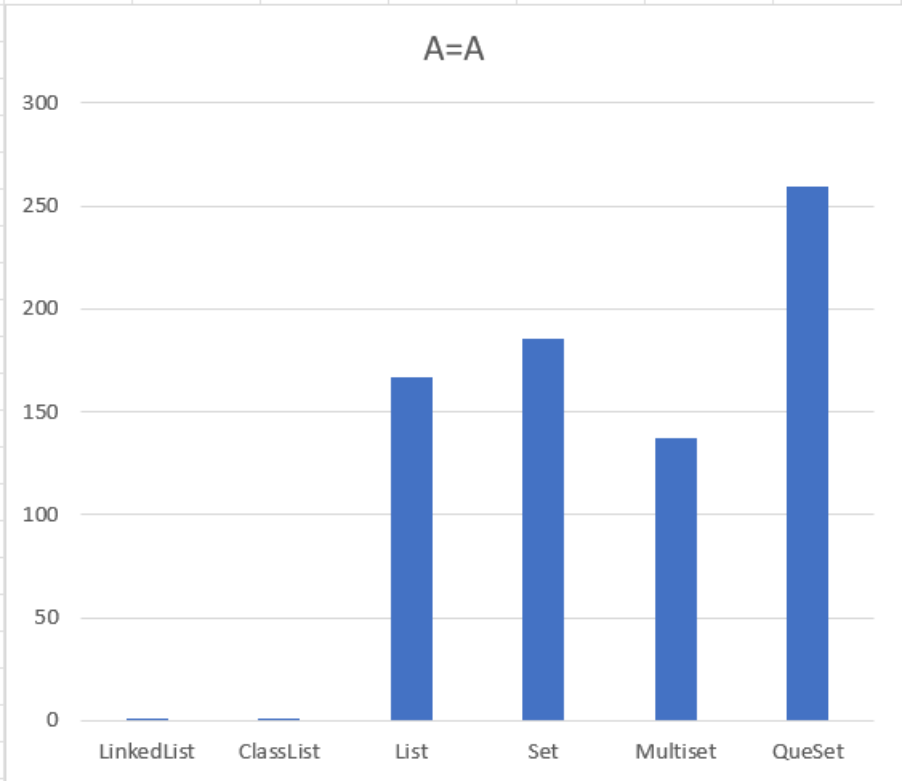

Рисунок 22 - Результат работы операции “Равенство A=A”

Операция равенства множеств в очередной раз демонстрирует быстродействие структур данных, работающих с односвязным списком.

Название	LinkedList	ClassList	List	Set	Multiset	QueSet
A=B	0,2332	0,0068	1,1965	1,7436	1,5409	1,733

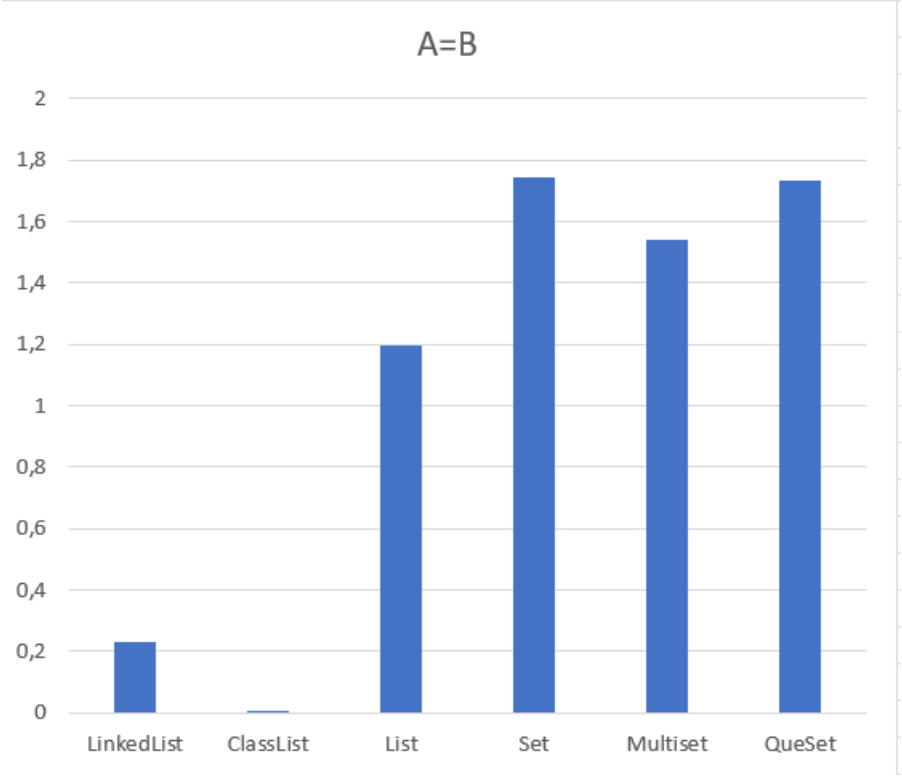

Рисунок 23 - Результат работы операции “Равенство A=B”

Выводы из операции проверки равенства A и B соответствуют выводам, сделанным на основе операции равенства A и A.

Название	LinkedList	ClassList	List	Set	Multiset	QueSet
Объединение	0,5839	1,3581	277,6382	33,5114	281,14	430,8273

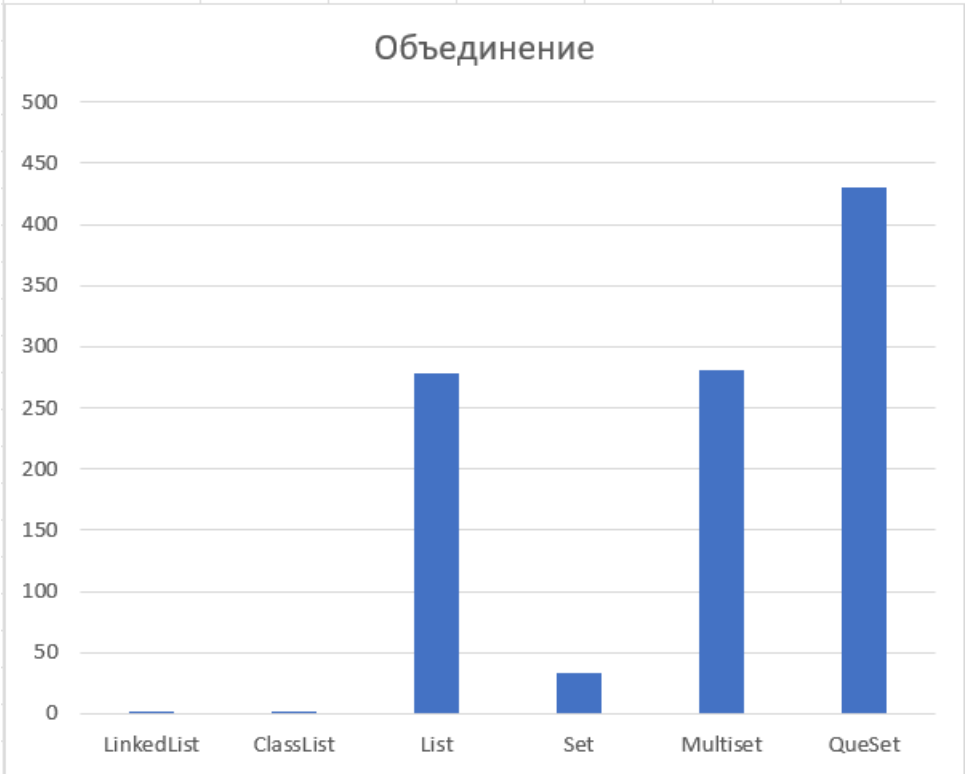

Рисунок 24- Результат работы операции “Объединение”

Результаты операции “Объединения” соответствуют средним результатам по остальным операциям. Единственно, стоит отметить быстроту контейнерного класса на основе set.

Название	LinkedList	ClassList	List	Set	Multiset	QueSet
Пересечение	0,721	0,5961	147,8306	177,1135	149,2396	197,7599

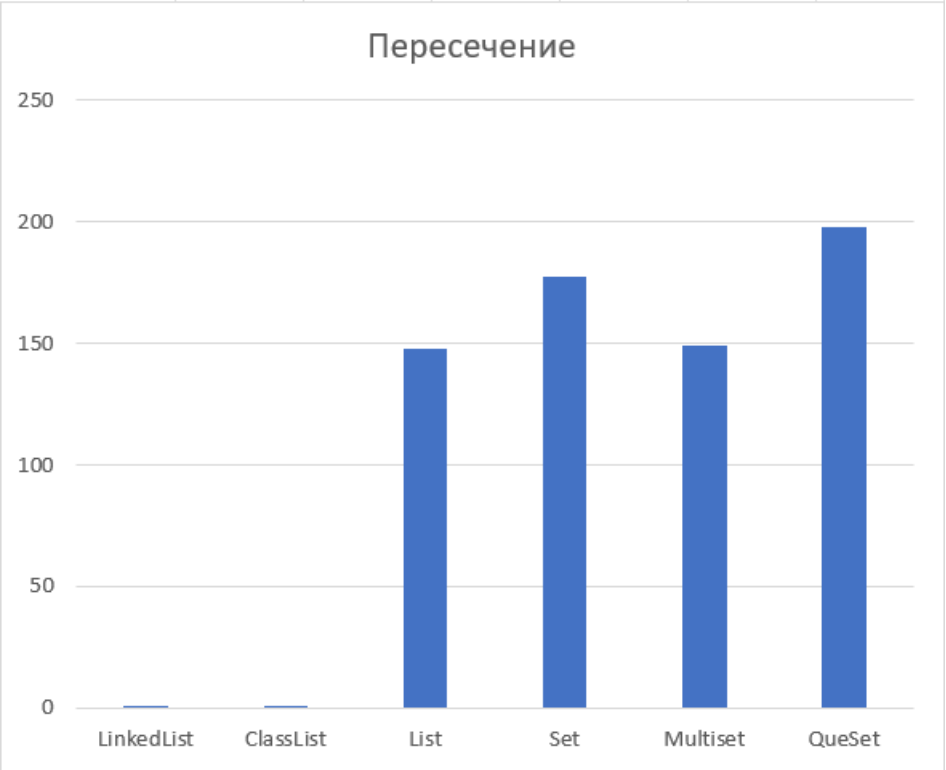

Рисунок 25 - Результат работы операции “Пересечение”

В операции “Пересечение”, как в прочем и большинстве других худшие результаты показал контейнерный класс на основе очереди.

Название	LinkedList	ClassList	List	Set	Multiset	QueSet
Разность A-B	0,2273	0,9435	199,4644	185,7681	197,642	271,8465

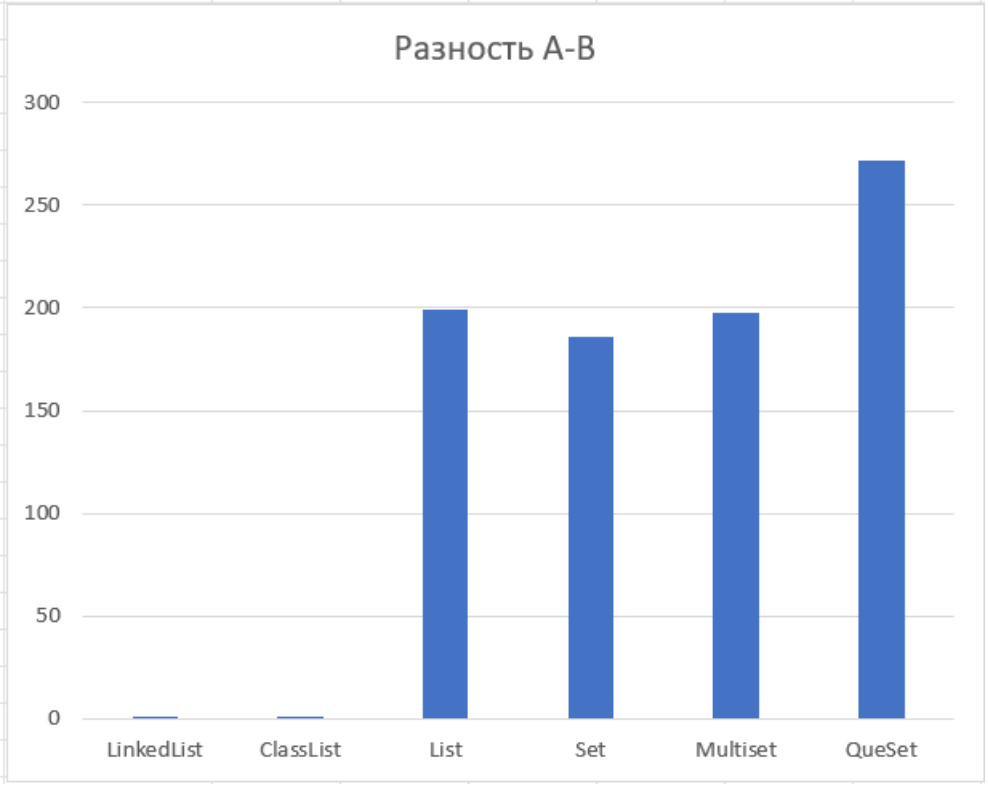

Рисунок 26 - Результат работы операции “Разность A-B”

Быстродействие контейнерных классов в операции разность довольно схожи, за исключением класса QueSet.

Название	LinkedList	ClassList	List	Set	Multiset	QueSet
Разность B-A	0,2224	0,9565	219,7198	189,5079	209,0036	284,1704

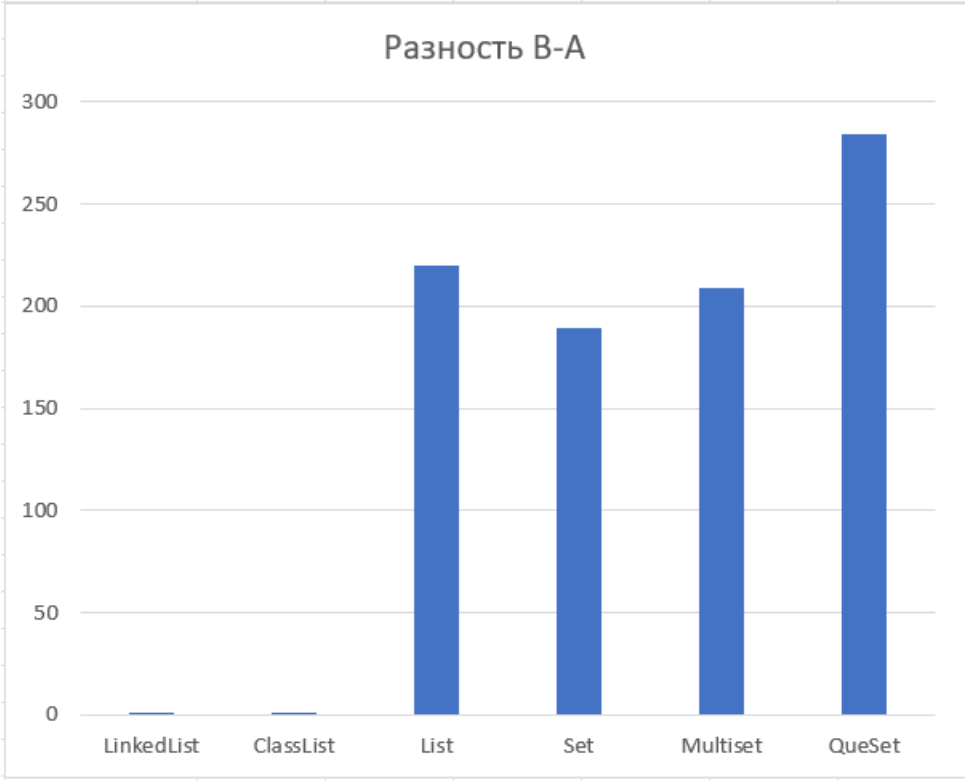

Рисунок 27 - Результат работы операции “Разность B-A”

Результаты аналогичны предыдущей операции

Название	LinkedList	ClassList	List	Set	Multiset	QueSet
Симметричная	1,5005	2,3956	699,033	221,0879	650,5302	948,0178

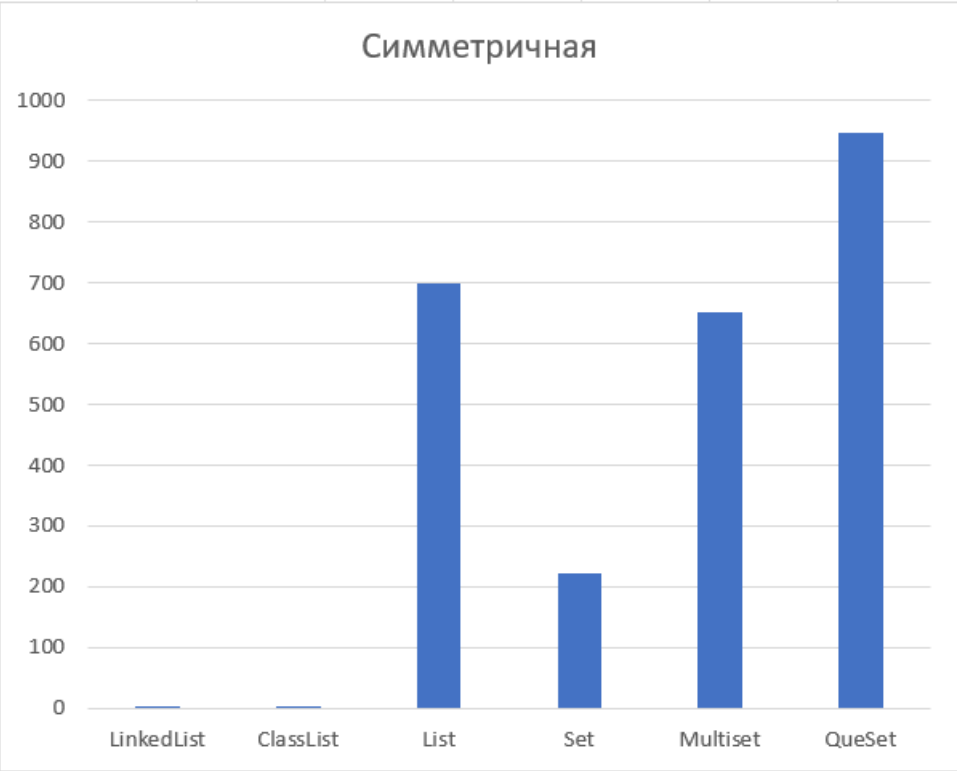

Рисунок 28 - Результат работы операции “Симметричная разность”

“Симметричная разность” – самая показательная операция, так как является самой сложной из всех присутствующих. Она включает в себя такие операции как разность, объединение и пересечение. Структуры данных, в основе которых присутствует односвязный список окончательно зарекомендовали себя, как самые быстро действенные. Также довольно неплохие результаты у класса на основе set. Класс очередь ожидаемо справился хуже всех.

Заключение

В данной курсовой работе была поставлена задача сравнения времени выполнения основных операций над множеством различными реализациями структуры данных "Множество". Для ее реализации было произведено решение подзадач: постановка задачи и анализ предметной области в том числе: определены основные понятие и определения, проанализированы требования к интерфейсу пользователя, к структурам данных, к программным средствам. Реализовано проектирование программы, кодирования, представлены диаграммы компонентов. Так же было проведено функциональное тестирование. Наконец были выполнены итоговые замеры, на основе которых были сделаны следующие выводы: для работы со множествами лучше всего использовать класс, основой для которого является односвязный список, так как в среднем он занимает второе место по быстро действию и первое по стабильности среди всех представленных структур данных, так же при работе с ним можно использовать все преимущества, которые нам предоставляет концепция ООП. В свою очередь класс на основе очереди вряд ли стоит использовать для работы со множествами из-за своей неэффективности по времени и сложной реализации, не предоставляющей возможности простого прохода по структуре данных.

Список использованных источников

1. Т. А. Павловская. C/C++. Программирование на языке высокого уровня. – СПб.: Питер, 2003. – 461 с
2. Список list в C++: полный материал – Статья // [Электронный ресурс]. URL: https://codelessons.ru/cplusplus/spisok-list-v-s-polnyj-material.html?ysclid=lf8m5ey2fj493877934#2
3. Microsoft Learn Документация. Класс set. – Статья // [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/cpp/standard-library/set-class?view=msvc-140
4. Microsoft Learn Документация. Класс Multiset. – Статья // [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/cpp/standard-library/ multiset-class?view=msvc-170
5. Microsoft Learn Документация. Класс queue. – Статья // [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/cpp/standard-library/ queue-class?view=msvc-170
6. Односвязный линейный список – Статья // [Электронный ресурс]. URL: https://prog-cpp.ru/data-ols/?ysclid=lf8lpkdhdk882230434

Приложение А.

КОД ПРОГРАММЫ

(обязательное)

ZXCursss.cpp

```
// ZXCursss.cpp : Этот файл содержит функцию "main". Здесь начинается и
заканчивается выполнение программы.
//

#include <iostream>
#include "Multiset.h"
#include "QueSet.h"
#include "ClassHeader.h"
#include "Linkedlist.h"
#include "List.h"
#include "Set.h"
#include <chrono>
#include <iomanip>
#include <functional>

template <typename T, typename... Args>
double measure_time(T&& func, Args &&... args) {
    auto start = std::chrono::steady_clock::now();
    std::invoke(std::forward<T>(func), std::forward<Args>(args)...);
    auto end = std::chrono::steady_clock::now();
    return std::chrono::duration<double, std::milli>(end - start).count();
}

int main()
{
    listik listikA;
    listik listikB;

    listok listokA;
    listok listokB;

    PlentyList PlentyListA;
    PlentyList PlentyListB;

    Plenty SetA;
    Plenty SetB;

    Multiset multisetA;
    Multiset multisetB;

    QueSet QueSetA;
    QueSet QueSetB;

    const int ROWS = 12;
    const int COLS = 7;
    int N;

    while (true) {

        cin >> N;
        if (N < 500 or N>3000) {
            cout << "Не корректный ввод множетсва, введите значение от 500 и до
3000" << endl;
        }
        else {
            break;
        }
    }
        string table[ROWS][COLS];

    for (int i = 0; i < ROWS; i++) {//инициализируем нулями
        for (int j = 0; j < COLS; j++) {
            table[i][j] = "0";
        }
    }
```

```
    table[0][0] = "Название     ";
    table[0][1] = "LinkedList   ";
    table[0][2] = "ClassList    ";
    table[0][3] = "List         ";
    table[0][4] = "Set          ";
    table[0][5] = "Multiset     ";
    table[0][6] = "QueSet       ";

    table[1][0] = "Создание     ";
    table[2][0] = "Мощность     ";
    table[3][0] = "Подм-во A и A";
    table[4][0] = "Подм-во B и A";
    table[5][0] = "A=A          ";
    table[6][0] = "A=B          ";
    table[7][0] = "Об-ие A и B  ";
    table[8][0] = "Пер-ние A и B";
    table[9][0] = "Раз-ть A-B   ";
    table[10][0] ="Раз-ть B-A   ";
    table[11][0] ="Симмет       ";


    listikB.create(N, 0, 10 * N);
    table[1][1] = to_string(measure_time(&listik::create, &listikA, N, 0, 10 *
N));
    table[2][1] = to_string(measure_time(&listik::count, &listikA,
listikA.first));
    table[3][1] = to_string(measure_time(&listik::is_subset, &listikA,
listikA.first,listikA.first));
    table[4][1] = to_string(measure_time(&listik::is_subset, &listikA,
listikA.first, listikB.first));
    table[5][1] = to_string(measure_time(&listik::is_equal, &listikA,
listikA.first, listikA.first));
    table[6][1] = to_string(measure_time(&listik::is_equal, &listikA,
listikA.first, listikB.first));
    table[7][1] = to_string(measure_time(&listik::combine, &listikA,
listikA.first, listikB.first));
    table[8][1] = to_string(measure_time(&listik::intersection, &listikA,
listikA.first, listikB.first));
    table[9][1] = to_string(measure_time(&listik::difference, &listikA,
listikA.first, listikB.first));
    table[10][1] = to_string(measure_time(&listik::difference, &listikA,
listikB.first, listikA.first));
    table[11][1] = to_string(measure_time(&listik::simmetric, &listikA,
listikA.first, listikB.first));


    listokB.create(N, 0, 10 * N, 0);
    table[1][2] = to_string(measure_time(&listok::create, &listokA, N, 0, 10 * N,
0));
    table[2][2] = to_string(measure_time(&listok::count, &listokA,
listokA.first));
    table[3][2] = to_string(measure_time(&listok::is_subset, &listokA,
listokA.first, listokA.first));
    table[4][2] = to_string(measure_time(&listok::is_subset, &listokA,
listokA.first, listokB.first));
    table[5][2] = to_string(measure_time(&listok::is_equal, &listokA,
listokA.first, listokA.first));
    table[6][2] = to_string(measure_time(&listok::is_equal, &listokA,
listokA.first, listokB.first));
    table[7][2] = to_string(measure_time(&listok::combine, &listokA,
listokA.first, listokB.first));
    table[8][2] = to_string(measure_time(&listok::intersection, &listokA,
listokA.first, listokB.first));
    table[9][2] = to_string(measure_time(&listok::difference, &listokA,
listokA.first, listokB.first));
    table[10][2] = to_string(measure_time(&listok::difference, &listokA,
```

```
listokB.first, listokA.first));
    table[11][2] = to_string(measure_time(&listok::simmetric, &listokA,
listokA.first, listokB.first));

    PlentyListB.create(N, 0, 10 * N, 0);
    table[1][3] = to_string(measure_time(&PlentyList::create, &PlentyListA, N,
0,10*N,0));
    table[2][3] = to_string(measure_time(&PlentyList::count, &PlentyListA));
    table[3][3] = to_string(measure_time(&PlentyList::is_subset,
&PlentyListA,PlentyListA.lst,PlentyListA.lst));
    table[4][3] = to_string(measure_time(&PlentyList::is_subset, &PlentyListA,
PlentyListA.lst, PlentyListB.lst));
    table[5][3] = to_string(measure_time(&PlentyList::is_equal, &PlentyListA,
PlentyListA.lst, PlentyListA.lst));
    table[6][3] = to_string(measure_time(&PlentyList::is_equal, &PlentyListA,
PlentyListA.lst, PlentyListB.lst));
    table[7][3] = to_string(measure_time(&PlentyList::combine, &PlentyListA,
PlentyListA.lst, PlentyListB.lst));
    table[8][3] = to_string(measure_time(&PlentyList::intersection, &PlentyListA,
PlentyListA.lst, PlentyListB.lst));
    table[9][3] = to_string(measure_time(&PlentyList::difference, &PlentyListA,
PlentyListA.lst, PlentyListB.lst));
    table[10][3] = to_string(measure_time(&PlentyList::difference, &PlentyListA,
PlentyListB.lst, PlentyListA.lst));
    table[11][3] = to_string(measure_time(&PlentyList::simmetric, &PlentyListA,
PlentyListA.lst, PlentyListB.lst));

    SetB.create(N, 0, 10 * N, 0);
    table[1][4] = to_string(measure_time(&Plenty::create, &SetA, N, 0, 10 * N,
0));
    table[2][4] = to_string(measure_time(&Plenty::count, &SetA));
    table[3][4] = to_string(measure_time(&Plenty::is_subset,
&SetA,SetA.lst,SetA.lst));
    table[4][4] = to_string(measure_time(&Plenty::is_subset, &SetA, SetA.lst,
SetB.lst));
    table[5][4] = to_string(measure_time(&Plenty::is_equal, &SetA, SetA.lst,
SetA.lst));
    table[6][4] = to_string(measure_time(&Plenty::is_equal, &SetA, SetA.lst,
SetB.lst));
    table[7][4] = to_string(measure_time(&Plenty::combine, &SetA, SetA.lst,
SetB.lst));
    table[8][4] = to_string(measure_time(&Plenty::intersection, &SetA, SetA.lst,
SetB.lst));
    table[9][4] = to_string(measure_time(&Plenty::difference, &SetA, SetA.lst,
SetB.lst));
    table[10][4] = to_string(measure_time(&Plenty::difference, &SetA, SetB.lst,
SetA.lst));
    table[11][4] = to_string(measure_time(&Plenty::simmetric, &SetA, SetA.lst,
SetB.lst));


    multisetB.create(0, 10 * N, N);
    table[1][5] = to_string(measure_time(&Multiset::create, &multisetA, 0, 10 * N,
N));
    table[2][5] = to_string(measure_time(&Multiset::count, &multisetA));
    table[3][5] = to_string(measure_time(&Multiset::is_subset,
&multisetA,multisetA.s,multisetA.s));
    table[4][5] = to_string(measure_time(&Multiset::is_subset, &multisetA,
multisetA.s, multisetB.s));
    table[5][5] = to_string(measure_time(&Multiset::is_equal, &multisetA,
multisetA.s, multisetA.s));
    table[6][5] = to_string(measure_time(&Multiset::is_equal, &multisetA,
multisetA.s, multisetB.s));
    table[7][5] = to_string(measure_time(&Multiset::combine, &multisetA,
multisetA.s, multisetB.s));
    table[8][5] = to_string(measure_time(&Multiset::intersection, &multisetA,
multisetA.s, multisetB.s));
```

```
    table[9][5] = to_string(measure_time(&Multiset::difference, &multisetA,
multisetA.s, multisetB.s));
    table[10][5] = to_string(measure_time(&Multiset::difference, &multisetA,
multisetB.s, multisetA.s));
    table[11][5] = to_string(measure_time(&Multiset::simmetric, &multisetA,
multisetA.s, multisetB.s));


    QueSetB.create(0, 10 * N, N);
    table[1][6] = to_string(measure_time(&QueSet::create, &QueSetA, 0, 10 * N,
N));
    table[2][6] = to_string(measure_time(&QueSet::count, &QueSetA));
    table[3][6] = to_string(measure_time(&QueSet::is_subset,
&QueSetA,QueSetA.s,QueSetA.s));
    table[4][6] = to_string(measure_time(&QueSet::is_subset, &QueSetA, QueSetA.s,
QueSetB.s));
    table[5][6] = to_string(measure_time(&QueSet::is_equal, &QueSetA, QueSetA.s,
QueSetA.s));
    table[6][6] = to_string(measure_time(&QueSet::is_equal, &QueSetA, QueSetA.s,
QueSetB.s));
    table[7][6] = to_string(measure_time(&QueSet::combine, &QueSetA, QueSetA.s,
QueSetB.s));
    table[8][6] = to_string(measure_time(&QueSet::intersection, &QueSetA,
QueSetA.s, QueSetB.s));
    table[9][6] = to_string(measure_time(&QueSet::difference, &QueSetA, QueSetA.s,
QueSetB.s));
    table[10][6] = to_string(measure_time(&QueSet::difference, &QueSetA,
QueSetB.s, QueSetA.s));
    table[11][6] = to_string(measure_time(&QueSet::simmetric, &QueSetA, QueSetA.s,
QueSetB.s));

    // Print table rows
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            cout << setw(13) <<right<< table[i][j]; // устанавливаем ширину 13
символов для каждой ячейки таблицы
        }
        cout << endl;
    }


}
```

ClassHeader.h

```
#pragma once
#include <iostream>
#include <string>
using namespace std;


struct Node {
      int val;
      Node* next;
      Node(int _val) : val(_val), next(nullptr) {}
};

class listok {
public:
      Node* first;
      listok() {
            this->first = NULL;
      }
      listok(Node* head) {
            this->first = head;
      }
      bool empty(Node* headptr);
```

```
        bool not_belong_to(Node* p, int _val);
        bool is_subset(Node* heada, Node* headb);
        bool is_equal(Node* heada, Node* headb);
        int count(Node* headptr);
        string printstr(Node* headptr, string c);
        Node* create(int x, int min, int max, int param);
        Node* deletelist(Node*& headptr);
        Node* add_to_begin(Node*& p, int _val);

        listok* combine(Node* heada, Node* headb);
        listok* intersection(Node* heada, Node* headb);
        listok* difference(Node* heada, Node* headb);
        listok* simmetric(Node* heada, Node* headb);
        ~listok() {
                cout << "蔓珙?膩糖痼牝铕? Деструктор" << endl;
                auto iter = first;
                while (iter != nullptr) {
                        first = iter->next;
                        delete iter;
                        iter = first;
                }
        };

};
```

Linkedlist.h

```
#pragma once
#include <iostream>
#include <string>
using namespace std;


struct LinkedNode {
        int val;
        LinkedNode* next;
        LinkedNode(int _val) : val(_val), next(nullptr) {}
};

struct listik {
        LinkedNode* first;
        LinkedNode* last;
        listik() : first(nullptr), last(nullptr) {}
        LinkedNode* create(int x, int min, int max);
        bool empty(LinkedNode* headptr);
        bool not_belong_to(LinkedNode* p, int _val);
        bool is_subset(LinkedNode* heada, LinkedNode* headb);
        bool is_equal(LinkedNode* heada, LinkedNode* headb);
        int count(LinkedNode* headptr);
        string printstr(LinkedNode* headptr, string c);
        LinkedNode* deletelist(LinkedNode*& headptr);
        LinkedNode* add_to_begin(LinkedNode*& p, int _val);
        LinkedNode* combine(LinkedNode* heada, LinkedNode* headb);
        LinkedNode* intersection(LinkedNode* heada, LinkedNode* headb);
        LinkedNode* difference(LinkedNode* heada, LinkedNode* headb);
        LinkedNode* simmetric(LinkedNode* heada, LinkedNode* headb);
};
```

List.h

```
#pragma once
#include <list>
#include <string>
//using namespace std;
```

```
class PlentyList {
public:
	PlentyList() {
		std::list<int> lst;
	}
	std::list <int> lst;
	bool is_empty(std::list<int> lst);
	void push_up(int val);
	std::string print(std::list<int> lst, char c);
	bool is_unique(std::list<int> lst, int val);
	int count();
	void create(int x, int min, int max, int last);
	bool is_subset(std::list<int> lsta, std::list<int> lstb);
	bool is_equal(std::list<int> lsta, std::list<int> lstb);
	PlentyList combine(std::list<int> lsta, std::list<int> lstb);
	PlentyList intersection(std::list<int> lsta, std::list<int> lstb);
	PlentyList difference(std::list<int> lsta, std::list<int> lstb);
	PlentyList simmetric(std::list<int> lsta, std::list<int> lstb);
};
```

Multiset.h

```
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <set>
#include <random>

using namespace std;

class Multiset {
public:
    multiset<int> s;

    //Multiset();


    int count();
    bool is_unique(multiset<int>ss,int value);
    void push_up(int value);
    bool is_empty();
    void create(int min, int max, int count);
    string printstr(multiset<int> s, string c);
    bool is_subset(std::multiset <int> lsta, std::multiset <int> lstb);
    bool is_equal(std::multiset <int> lsta, std::multiset <int> lstb);
    Multiset combine(std::multiset <int> lsta, std::multiset <int> lstb);
    Multiset intersection(std::multiset <int> lsta, std::multiset <int> lstb);
    Multiset difference(std::multiset <int> lsta, std::multiset <int> lstb);
    Multiset simmetric(std::multiset <int> lsta, std::multiset <int> lstb);



};
```

QueSet.h

```
#pragma once
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <queue>
#include <random>
```

```
using namespace std;

class QueSet {
public:
    queue<int> s;

    //Multiset();


    int count();
    bool is_unique(queue<int>ss, int value);
    void push_up(int value);
    bool is_empty();
    void create(int min, int max, int count);
    string printstr(queue<int> s, string c);
    bool is_subset(std::queue <int> lsta, std::queue <int> lstb);
    bool is_equal(std::queue <int> lsta, std::queue <int> lstb);
    QueSet combine(std::queue <int> lsta, std::queue <int> lstb);
    QueSet intersection(std::queue <int> lsta, std::queue <int> lstb);
    QueSet difference(std::queue <int> lsta, std::queue <int> lstb);
    QueSet simmetric(std::queue <int> lsta, std::queue <int> lstb);



};
```

Set.h

```
#pragma once
#include <set>
#include <string>
using namespace std;
class Plenty {
public:
      Plenty() {
            set<int> lst;
      }
      set <int> lst;
      bool is_empty(set<int> lst);
      void push_up(int val);
      string print(set<int> lst, char c);
      bool is_unique(set<int> lst, int val);
      int count();
      void create(int x, int min, int max, int last);
      bool is_subset(set<int> lsta, set<int> lstb);
      bool is_equal(set<int> lsta, set<int> lstb);
      Plenty combine(set<int> lsta, set<int> lstb);
      Plenty intersection(set<int> lsta, set<int> lstb);
      Plenty difference(set<int> lsta, set<int> lstb);
      Plenty simmetric(set<int> lsta, set<int> lstb);
};
```

ClassSource.cpp

```
#include "ClassHeader.h";
#include <stdlib.h>
#include <time.h>
#include <random>


Node* listok::add_to_begin(Node*& headptr, int _val) {
    if (!not_belong_to(headptr, _val)) {
        return headptr;
```

```
    }
    Node* p1 = new Node(_val);

    p1->next = headptr;
    headptr = p1;
    return headptr;
};

int GetRandomNumber(int min, int max)
{
    srand(time(NULL));
    int num = min + rand() % (max - min + 1);
    return num;
};

Node* listok::create(int x, int min, int max, int param) {
    if (min >= max) {
        return nullptr;
    }
    if (max - min < x) {
        return nullptr;
    }

    for (int i = 0; i < x; i++) {
        int value = min + rand() % (max - min + 1);
        while (value % 10 <= param) {
            value = min + rand() % (max - min + 1);
        }
        while (!this->not_belong_to(this->first, value)) {
            value = min + rand() % (max - min + 1);
        }
        this->add_to_begin(this->first, value);
    }
    return first;
};

Node* listok::deletelist(Node*& headptr) {
    if (empty(headptr)) {
        return headptr;
    }
    Node* p = headptr;
    while (p) {
        p = p->next;
        delete headptr;
        headptr = p;
    }
    return headptr;
};

bool listok::empty(Node* headptr) {
    return headptr == nullptr;
}

int listok::count(Node* headptr) {
    int i = 0;
    if (empty(headptr)) return 0;
    Node* p = headptr;
    while (p) {
        p = p->next;
        i++;
    }
    return i;
};

string listok::printstr(Node* headptr, string c) {
    string res;
```

```
    if (empty(headptr)) {
        return " ";
    }
    while (headptr) {
        int xx = headptr->val;

        res += to_string(xx) + c;
        headptr = headptr->next;
    }
    res.pop_back();
    return res;
};

bool listok::not_belong_to(Node* p, int _val) {
    if (empty(p)) {
        return true;
    }
    while (p) {
        if (p->val == _val) {
            return false;
        }
        p = p->next;
    }
    return true;
};

//2 鬣糖?

bool listok::is_subset(Node* heada, Node* headb) {//□怆□驿? 腓 镱潇眍弩糖忸?
    if (count(heada) > count(headb)) {
        return false;
    }
    if (empty(heada) || empty(headb)) {
        return false;
    }
    while (heada) {
        if (not_belong_to(headb, heada->val)) {
            return false;
        }
        heada = heada->next;

    }
    return true;
};

bool listok::is_equal(Node* heada, Node* headb) {//疣忏?腓 祉铈驿疋?
    if (empty(heada) || empty(headb)) {
        return true;
    }
    if ((count(heada) == count(headb)) && is_subset(heada, headb)) {
        return true;
    }
    return false;
}

listok* listok::combine(Node* heada, Node* headb) {//钺□滂礤龛?祉铈羼蜮
    auto List = new listok{};
    while (heada) {
        add_to_begin(List->first, heada->val);
        heada = heada->next;
    }
    while (headb) {
        add_to_begin(List->first, headb->val);
        headb = headb->next;
    }
    return List;
};
```

```
/*Node* list::combine2(Node* heada, Node* headb) {//钺□滂磙龛?祉铈羼蜮
    list lst;
    while (heada) {
        add_to_begin(lst.first, heada->val);
        heada = heada->next;
    }
    while (headb) {
        add_to_begin(lst.first, headb->val);
        headb = headb->next;
    }
    return lst.first;
};*/

/*Node* list::intersection2(Node* heada, Node* headb) {//镥疱皴麋龛?祉铈羼蜮
    list lst;
    while (heada) {
        if (!not_belong_to(headb, heada->val)) {
            add_to_begin(lst.first, heada->val);
        }
        heada = heada->next;
    }
    return lst.first;
};*/

listok* listok::intersection(Node* heada, Node* headb) {//镥疱皴麋龛?祉铈羼蜮
    auto Listok = new listok{};
    while (heada) {
        if (!not_belong_to(headb, heada->val)) {
            add_to_begin(Listok->first, heada->val);
        }
        heada = heada->next;
    }
    return Listok;
};

listok* listok::difference(Node* heada, Node* headb) {
    auto List = new listok{};
    while (heada) {
        if (not_belong_to(headb, heada->val)) {
            add_to_begin(List->first, heada->val);
        }
        heada = heada->next;
    }
    return List;

};

listok* listok::simmetric(Node* heada, Node* headb) {
    //auto List = new list{};
    ///lst.first = difference(combine(heada, headb), intersection(heada, headb));
    //return difference(combine2(heada, headb), intersection2(heada, headb));
    auto List = new listok{};
    Node* tmpa = heada;
    while (heada) {
        if (not_belong_to(headb, heada->val)) {
            add_to_begin(List->first, heada->val);
        }
        heada = heada->next;
    }
    while (headb) {
        if (not_belong_to(tmpa, headb->val)) {
            add_to_begin(List->first, headb->val);
        }
        headb = headb->next;
    }
    return List;
```

```
}
```

LinkedSource.cpp

```
#include "Linkedlist.h";
#include <stdlib.h>
#include <time.h>
#include <random>


LinkedNode* listik::add_to_begin(LinkedNode*& headptr, int _val) {
    if (!not_belong_to(headptr, _val)) {
        return headptr;
    }
    LinkedNode* p1 = new LinkedNode(_val);

    p1->next = headptr;
    headptr = p1;
    return headptr;
};

int GetRandomNumber2(int min, int max)
{
    srand(time(NULL));
    int num = min + rand() % (max - min + 1);
    return num;
};

LinkedNode* listik::create(int x, int min, int max) {
    if (min >= max) {
        return nullptr;
    }
    if (max - min < x) {
        return nullptr;
    }

    int rnd;
    LinkedNode* p = new LinkedNode(GetRandomNumber2(min, max));
    first = p;
    last = p;
    for (int i = 0; i < x - 1; ) {
        rnd = min + rand() % (max - min + 1);
        LinkedNode* tmp = first;
        add_to_begin(first, rnd);
        if (tmp != first) {
            i++;
        }

    }
    return first;
};

LinkedNode* listik::deletelist(LinkedNode*& headptr) {
    if (empty(headptr)) {
        return headptr;
    }
    LinkedNode* p = headptr;
    while (p) {
        p = p->next;
        delete headptr;
        headptr = p;
    }
    return headptr;
```

```
};

bool listik::empty(LinkedNode* headptr) {
    return headptr == nullptr;
}

int listik::count(LinkedNode* headptr) {
    int i = 0;
    if (empty(headptr)) return 0;
    LinkedNode* p = headptr;
    while (p) {
        p = p->next;
        i++;
    }
    return i;
};

string listik::printstr(LinkedNode* headptr, string c) {
    string res;

    if (empty(headptr)) {
        return " ";
    }
    while (headptr) {
        int xx = headptr->val;

        res += to_string(xx) + c;
        headptr = headptr->next;
    }
    res.pop_back();
    return res;
};

bool listik::not_belong_to(LinkedNode* p, int _val) {
    if (empty(p)) {
        return true;
    }
    while (p) {
        if (p->val == _val) {
            return false;
        }
        p = p->next;
    }
    return true;
};

//2 鬣糖?

bool listik::is_subset(LinkedNode* heada, LinkedNode* headb) {//□怆□驿? 腓 镱潇眍弩糖忸?
    if (count(heada) > count(headb)) {
        return false;
    }
    if (empty(heada) || empty(headb)) {
        return false;
    }
    while (heada) {
        if (not_belong_to(headb, heada->val)) {
            return false;
        }
        heada = heada->next;

    }
    return true;
};

bool listik::is_equal(LinkedNode* heada, LinkedNode* headb) {//疣忏?腓 祉铈驿疋?
```

```
    if (empty(heada) || empty(headb)) {
        return true;
    }
    if ((count(heada) == count(headb)) && is_subset(heada, headb)) {
        return true;
    }
    return false;
}

LinkedNode* listik::combine(LinkedNode* heada, LinkedNode* headb) {//钺□滂磙龛?祉铈羼蜮
    listik lst;
    while (heada) {
        add_to_begin(lst.first, heada->val);
        heada = heada->next;
    }
    while (headb) {
        add_to_begin(lst.first, headb->val);
        headb = headb->next;
    }
    return lst.first;
};

LinkedNode* listik::intersection(LinkedNode* heada, LinkedNode* headb) {//镥疱皴麋龛?祉铈羼蜮
    listik lst;
    while (heada) {
        if (!not_belong_to(headb, heada->val)) {
            add_to_begin(lst.first, heada->val);
        }
        heada = heada->next;
    }
    return lst.first;
};

LinkedNode* listik::difference(LinkedNode* heada, LinkedNode* headb) {
    listik lst;
    while (heada) {
        if (not_belong_to(headb, heada->val)) {
            add_to_begin(lst.first, heada->val);
        }
        heada = heada->next;
    }
    return lst.first;

};

LinkedNode* listik::simmetric(LinkedNode* heada, LinkedNode* headb) {
    listik lst;
    lst.first = difference(combine(heada, headb), intersection(heada, headb));
    return lst.first;
}
```

ListSource.cpp

```
#include "List.h"
void PlentyList::push_up(int val) {
      if (this->is_unique(this->lst, val)) {
            this->lst.emplace_front(val);
      }
};
std::string PlentyList::print(std::list<int> lst, char c) {
      if (lst.empty()) {
            return " ";
      }
```

```
	std::list<int>::iterator it;
	std::string res;
	for (it = lst.begin(); it != lst.end(); ++it) {
		res += std::to_string(*it);
		res += c;
	}
	res.pop_back();
	return res;
}
bool PlentyList::is_empty(std::list<int> lst) {
	return lst.empty();
}
bool PlentyList::is_unique(std::list<int> lst, int val) {
	std::list<int>::iterator it;

	for (it = lst.begin(); it != lst.end(); ++it) {
		if (*it == val) {
			return false;
		}
	}
	return true;
}
int PlentyList::count() {

	return lst.size();
}
void PlentyList::create(int x, int min, int max, int last) {
	for (int i = 0; i < x;) {
		int chislo = min + rand() % (max - min + 1);
		while (chislo % 10 <= last) {
			chislo = min + rand() % (max - min + 1);
		}
		while (!this->is_unique(this->lst, chislo)) {//цикл гарантия что
элемент добавится тк он будет уникальным
			chislo = min + rand() % (max - min + 1);
		}
		this->push_up(chislo);
		i++;
	}
}
bool PlentyList::is_subset(std::list<int> lsta, std::list<int> lstb) {
	if (lsta.size() > lstb.size()) {
		return false;
	}
	std::list<int>::iterator ita;
	std::list<int>::iterator itb;

	for (ita = lsta.begin(); ita != lsta.end(); ++ita) {
		if (is_unique(lstb, *ita)) {
			return false;
		}
	}
	return true;

}

bool PlentyList::is_equal(std::list<int> lsta, std::list<int> lstb) {
	if (lsta.size() != lstb.size()) {
		return false;
	}
	else {
		return is_subset(lsta, lstb);
	}
}
PlentyList PlentyList::combine(std::list<int> lsta, std::list<int> lstb) {
	PlentyList plenty;
	std::list<int>::iterator it;
```

```
	for (it = lsta.begin(); it != lsta.end(); ++it) {
		plenty.push_up(*it);
	}
	for (it = lstb.begin(); it != lstb.end(); ++it) {
		plenty.push_up(*it);
	}
	return plenty;

}
PlentyList PlentyList::intersection(std::list<int> lsta, std::list<int> lstb) {
	PlentyList plenty;
	std::list<int>::iterator it;
	for (it = lsta.begin(); it != lsta.end(); ++it) {
		if (!is_unique(lstb, *it)) {
			plenty.push_up(*it);
		}
	}
	return plenty;
}
PlentyList PlentyList::difference(std::list<int> lsta, std::list<int> lstb) {
	PlentyList plenty;
	std::list<int>::iterator it;
	for (it = lsta.begin(); it != lsta.end(); ++it) {
		if (is_unique(lstb, *it)) {
			plenty.push_up(*it);
		}
	}
	return plenty;
}
PlentyList PlentyList::simmetric(std::list<int> lsta, std::list<int> lstb) {
	//Plenty plenty;
	return difference(combine(lsta, lstb).lst, intersection(lsta, lstb).lst);
}
```

MultisetSource.cpp

```
#include "Multiset.h"
#include <string>
using namespace std;
int Multiset::count() {
	return s.size();
}
bool Multiset::is_unique(multiset<int>ss,int value) {
	return ss.find(value)==ss.end();
}
void Multiset::push_up(int value) {
	if (is_unique(this->s,value)) {
		s.insert(s.begin(), value);
	}
}
bool Multiset::is_empty() {
	return s.empty();
}
void Multiset::create(int min, int max, int count) {
	random_device rd;
	mt19937 gen(rd());
	uniform_int_distribution<int> dist(min, max);
	while (this->count() < count) {
		push_up(dist(gen));
	}
}
string Multiset::printstr(multiset<int>s, string c) {
	string res = "";
	for (auto it = s.begin(); it != s.end(); ++it) {
		res  += to_string(*it) + " ";;
		res += c;
		//cout << *it << " ";
```

```
	}
	res.pop_back();
	return res;

}
bool Multiset::is_subset(multiset <int> lsta, multiset <int> lstb) {
	if (lsta.size() > lstb.size()) {
		return false;
	}
	std::multiset<int>::iterator ita;
	std::multiset<int>::iterator itb;

	for (ita = lsta.begin(); ita != lsta.end(); ++ita) {
		if (is_unique(lstb, *ita)) {
			return false;
		}
	}
	return true;
}
bool Multiset::is_equal(multiset <int> lsta, multiset <int> lstb) {
	if (lsta.size() != lstb.size()) {
		return false;
	}
	else {
		return is_subset(lsta, lstb);
	}
}
Multiset Multiset::combine(std::multiset <int> lsta, std::multiset <int> lstb){
	Multiset plenty;
	std::multiset<int>::iterator it;
	for (it = lsta.begin(); it != lsta.end(); ++it) {
		plenty.push_up(*it);
	}
	for (it = lstb.begin(); it != lstb.end(); ++it) {
		plenty.push_up(*it);
	}
	return plenty;
}
Multiset Multiset::intersection(std::multiset <int> lsta, std::multiset <int>
lstb){
	Multiset plenty;
	std::multiset<int>::iterator it;
	for (it = lsta.begin(); it != lsta.end(); ++it) {
		if (!is_unique(lstb, *it)) {
			plenty.push_up(*it);
		}
	}
	return plenty;
}
Multiset Multiset::difference(std::multiset <int> lsta, std::multiset <int> lstb){
	Multiset plenty;
	std::multiset<int>::iterator it;
	for (it = lsta.begin(); it != lsta.end(); ++it) {
		if (is_unique(lstb, *it)) {
			plenty.push_up(*it);
		}
	}
	return plenty;
}
Multiset Multiset::simmetric(std::multiset <int> lsta, std::multiset <int> lstb){
	return difference(combine(lsta, lstb).s, intersection(lsta, lstb).s);
}
```

QueSetSource.cpp

```
#include "QueSet.h"
#include <string>
```

```
int QueSet::count() {
      return s.size();
}
bool QueSet::is_unique(queue<int>ss, int value) {
    queue<int> tempQueue = ss;
    while (!tempQueue.empty()) {
        if (tempQueue.front() == value) {
            return false;
        }
        tempQueue.pop();
    }
    return true;
}
void QueSet::push_up(int value) {
    if (is_unique(s,value)) {
        s.push(value);
    }
}
bool QueSet::is_empty() {
      return s.empty();
}
void QueSet::create(int min, int max, int count) {
    srand(time(NULL));  // установка seed для генерации случайных чисел
    while (this->count() < count) {
        int value = rand() % (max - min + 1) + min;
       this->push_up(value);
    }
}
string QueSet::printstr(queue<int> s, string c) {
    string result = "";
    queue<int> tempQueue = s;
    while (!tempQueue.empty()) {
        result += to_string(tempQueue.front()) + " ";
        tempQueue.pop();
    }
    return result;
}
bool QueSet::is_subset(std::queue <int> lsta, std::queue <int> lstb) {
    queue<int> otherQueue = lstb;
    queue<int> tempQueue =lsta;
    while (!tempQueue.empty()) {
        if (is_unique(lstb, tempQueue.front()) ){
            return false;
        }
        tempQueue.pop();
    }
    return true;
}
bool QueSet::is_equal(std::queue <int> lsta, std::queue <int> lstb) {
    if (lsta.size() != lstb.size()) {
        return false;
    }

    return is_subset(lsta,lstb);
}
QueSet QueSet::combine(std::queue <int> lsta, std::queue <int> lstb){
    QueSet result;
    queue<int> otherQueue = lstb;
    queue<int> tempQueue = lsta;
    while (!tempQueue.empty()) {
         result.push_up(tempQueue.front());
        tempQueue.pop();
    }
    while (!otherQueue.empty()) {
        result.push_up(otherQueue.front());
        otherQueue.pop();
    }
```

```
    return result;
}
QueSet QueSet::intersection(std::queue <int> lsta, std::queue <int> lstb){
    QueSet result;
    queue<int> otherQueue = lstb;
    queue<int> tempQueue = lsta;
    while (!tempQueue.empty()) {
        if (!is_unique(otherQueue, tempQueue.front())) {
            result.push_up(tempQueue.front());

        }
        tempQueue.pop();
    }
    return result;
}
QueSet QueSet::difference(std::queue <int> lsta, std::queue <int> lstb){
    QueSet result;
    queue<int> otherQueue = lstb;
    queue<int> tempQueue = lsta;
    while (!tempQueue.empty()) {
        if (is_unique(otherQueue, tempQueue.front())) {
            result.push_up(tempQueue.front());

        }
        tempQueue.pop();
    }
    return result;
}
QueSet QueSet::simmetric(std::queue <int> lsta, std::queue <int> lstb) {
    return difference(combine(lsta, lstb).s, intersection(lsta, lstb).s);
}
```

SetSource.cpp

```
#include "Set.h"
#include <stdlib.h>
#include <time.h>

void Plenty::push_up(int val) {

      this->lst.emplace(val);

};
string Plenty::print(set<int> lst, char c) {
      if (lst.empty()) {
            return " ";
      }
      set<int>::iterator it;
      string res;
      for (it = lst.begin(); it != lst.end(); ++it) {
            res += to_string(*it);
            res += c;
      }
      res.pop_back();
      return res;
}
bool Plenty::is_empty(set<int> lst) {
      return lst.empty();
}
bool Plenty::is_unique(set<int> lst, int val) {
      set<int>::iterator it;

      for (it = lst.begin(); it != lst.end(); ++it) {
```

```
			if (*it == val) {
				return false;
			}
		}
		return true;
}
int Plenty::count() {

		return lst.size();
}
void Plenty::create(int x, int min, int max, int last) {//x-мощность last-на какую
цифру должно заканчиваться число

		for (int i = 0; i < x;) {
			int chislo = min + rand() % (max - min + 1);
			while (chislo % 10 <= last) {
				chislo = min + rand() % (max - min + 1);
			}
			while (!this->is_unique(this->lst, chislo)) {//цикл гарантия что
элемент добавится тк он будет уникальным
				chislo = min + rand() % (max - min + 1);
			}
			this->push_up(chislo);
			i++;
		}

};
bool Plenty::is_subset(set<int> lsta, set<int> lstb) {
		if (lsta.size() > lstb.size()) {
			return false;
		}
		set<int>::iterator ita;
		set<int>::iterator itb;

		for (ita = lsta.begin(); ita != lsta.end(); ++ita) {
			if (is_unique(lstb, *ita)) {
				return false;
			}
		}
		return true;

}

bool Plenty::is_equal(set<int> lsta, set<int> lstb) {
		if (lsta.size() != lstb.size()) {
			return false;
		}
		else {
			return is_subset(lsta, lstb);
		}
}
Plenty Plenty::combine(set<int> lsta, set<int> lstb) {
		Plenty plenty;
		set<int>::iterator it;
		for (it = lsta.begin(); it != lsta.end(); ++it) {
			plenty.push_up(*it);
		}
		for (it = lstb.begin(); it != lstb.end(); ++it) {
			plenty.push_up(*it);
		}
		return plenty;

}
Plenty Plenty::intersection(set<int> lsta, set<int> lstb) {
		Plenty plenty;
		set<int>::iterator it;
		for (it = lsta.begin(); it != lsta.end(); ++it) {
```

```
			if (!is_unique(lstb, *it)) {
				plenty.push_up(*it);
			}
		}
		return plenty;
}
Plenty Plenty::difference(set<int> lsta, set<int> lstb) {
		Plenty plenty;
		set<int>::iterator it;
		for (it = lsta.begin(); it != lsta.end(); ++it) {
			if (is_unique(lstb, *it)) {
				plenty.push_up(*it);
			}
		}
		return plenty;
}
Plenty Plenty::simmetric(set<int> lsta, set<int> lstb) {
		//Plenty plenty;
		return difference(combine(lsta, lstb).lst, intersection(lsta, lstb).lst);
}
```