**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 Map»

ПГУ 09.03.04 – 04КР221.02 ПЗ

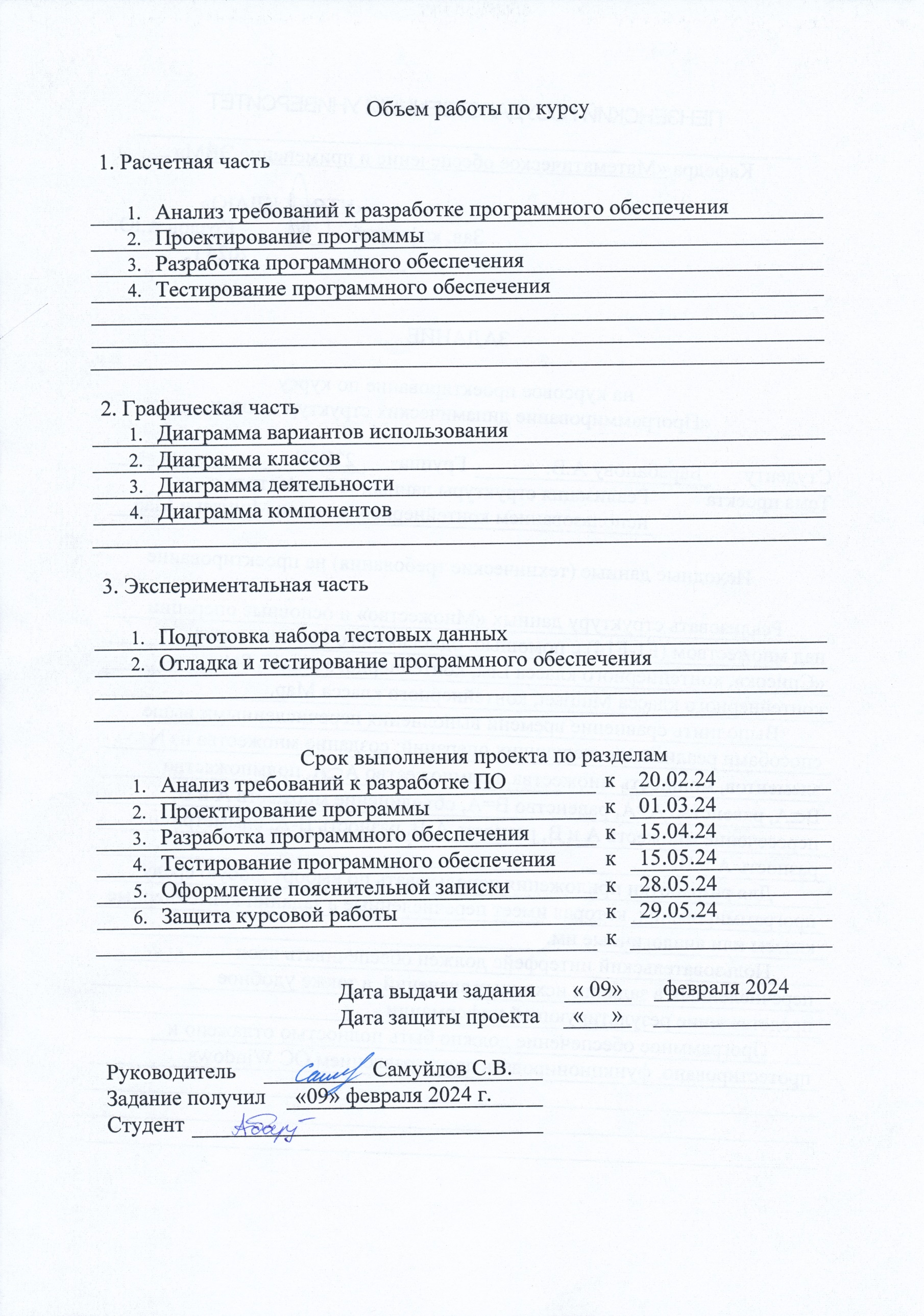
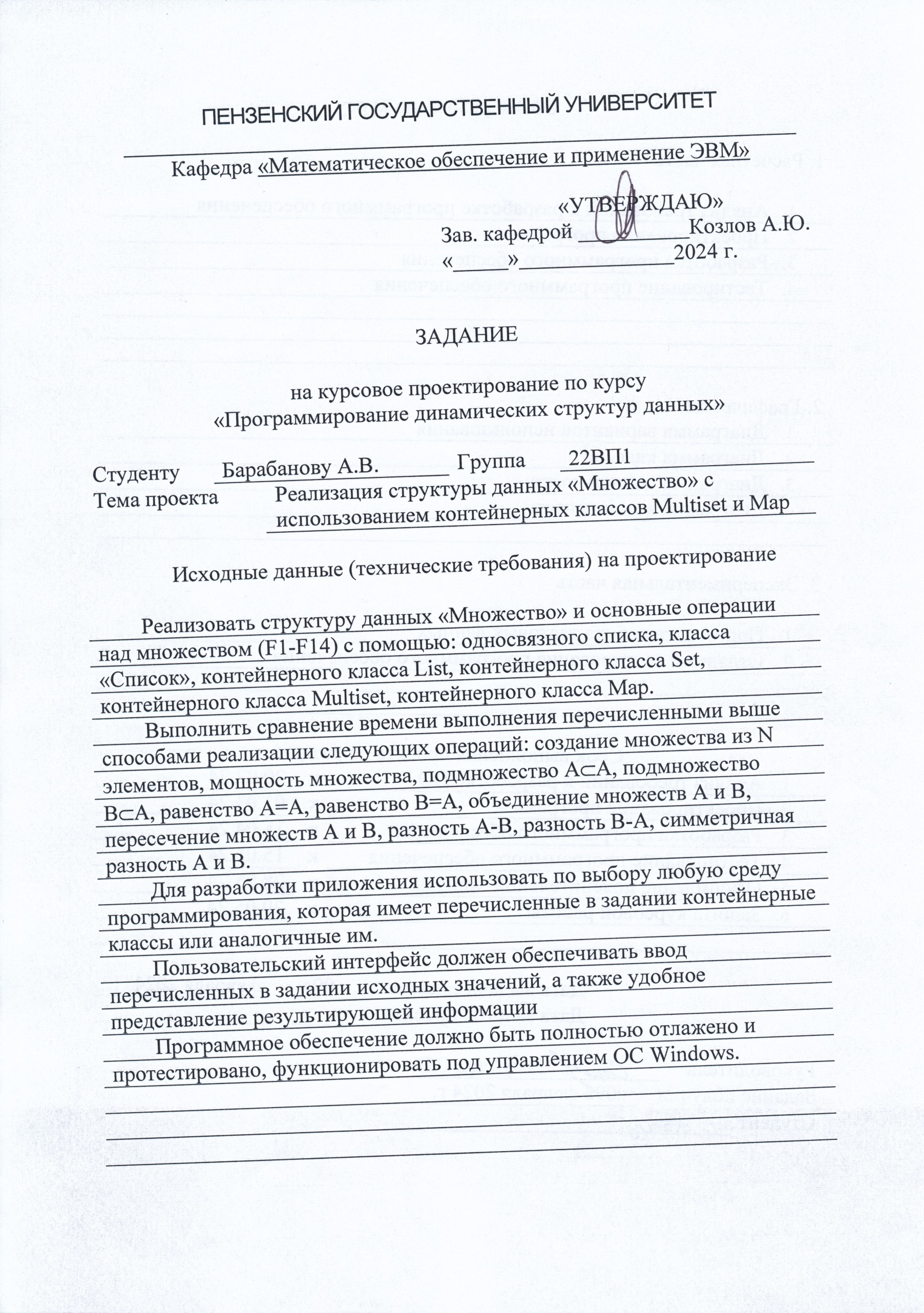
Направление подготовки – 09.03.04 Программная инженерия

Профиль подготовки – Программная инженерия

|  |  |  |
| --- | --- | --- |
| Выполнил студент |  | Барабанов А.В. |
| Группа |  | 22ВП1 |
|  |  |  |
| Руководитель |  |  |
| к.т.н., доцент |  | Самуйлов С.В. |

|  |  |
| --- | --- |
| Работа защищена с оценкой |  |
| Преподаватели |  |
|  |
| Дата защиты |  |

2024



**Реферат**

Пояснительная записка содержит 53 листа, 15 рисунков, 2 таблицы,   
 7 использованных источников, 2 приложения.

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

Объектом исследования является структура данных «Множество» и ее реализация различными способами.

Целью курсовой работы является разработка приложения для сравнения и анализа алгоритмов реализации основных операций над множествами.

Разработка проводилась на языке программирования С++ в среде программирования MS Visual Studio 2019.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | ПГУ 09.03.04 – 04КР221.02 ПЗ | | | | | |
|  |  |  |  |  |
| Изм. | Листт | № докум. | Подп. | Дата |
| Разраб. | | Барабанов А.В. |  |  | «Реализация структуры данных «Множество» с использованием контейнерных классов Multiset и Map»  Пояснительная записка. | Лит. | | | Лист | Листов |
| Пров. | | Самуйлов С.В. |  |  |  |  |  | 4 | 53 |
|  | |  |  |  | Группа 22ВП1 | | | | |
| Н. контр. | |  |  |  |
| Утв. | |  |  |  |

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

**Содержание**

[Введение 6](#_Toc162127782)

[1. Постановка задачи и анализ предметной области 7](#_Toc162127783)

[1.1. Основные понятия и определения 7](#_Toc162127784)

[1.2. Постановка задачи 8](#_Toc162127785)

[1.3. Анализ требований 8](#_Toc162127786)

[1.3.1. Требования к интерфейсу пользователя 8](#_Toc162127787)

[1.3.2. Требования к структурам данных 9](#_Toc162127788)

[1.3.3. Требования к программным средствам 11](#_Toc162127789)

[2. Проектирование программы 15](#_Toc162127790)

[2.1. Модель интерфейса 15](#_Toc162127791)

[2.2. Проектирование структур данных 15](#_Toc162127792)

[2.3. Структура программного обеспечения 16](#_Toc162127793)

# Введение

# Постановка задачи и анализ предметной области

## Основные понятия и определения

Множество – тип и структура данных, которое являются реализацией математического объекта множество. Данные типа множество позволяют хранить ограниченное число значений определенного типа без определенного порядка. Повторение значений, как правило, недопустимо.

Создание множества из N элементов – заполнение множества случайными значениями в количестве N элементов.

Мощность множества – текущее кол-во элементов во множестве.

Подмножество (A⊂B) – множество A является частью множества B.

Равенство (A=B) – множества A и B состоят из одинаковых элементов.

Объединение множеств (A∪B) – множество, состоящее из элементов, которые принадлежат или A, или B.

Пересечение множеств (A∩B) – множество, состоящее из элементов, которые принадлежат и A, и B.

Разность множеств (A\B) – множество, состоящее из элементов, которые принадлежат множеству A и не принадлежат B.

Симметричная разность множеств ((A∪B)\(A∩B)) – множество, включающее все элементы исходных множеств, не принадлежащие одновременно обоим исходным множествам.

Односвязный список – это линейная и однонаправленная структура данных, в которой данные сохраняются на узлах, и каждый узел связан ссылкой со своим следующим узлом. Каждый узел содержит поле данных и ссылку на следующий узел.

Контейнерный класс – это класс, предназначенный для хранения данных, организованных определённым образом. Один и тот же вид контейнера можно использовать для хранения данных различных типов.

Контейнер «List» – реализован как двусвязный список, то есть такой список, где каждый элемент имеет указатели на предыдущий и последовательный элемент.

Доступ к данным списка может быть двунаправленным и последовательным.

Контейнер «Set» – ассоциативный контейнер, который является контейнером переменного размера, поддерживающим эффективное получение значений элементов на основе значения соответствующего ключа. Кроме того, это простой ассоциативный контейнер, так как его значения элементов являются ключевыми значениями.

Используется для хранения и извлечения данных из коллекции (множества).

Контейнер «Multiset» – 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.

Контейнер «Map» – используется для хранения и извлечения данных из коллекции, в которой каждый элемент является парой, обладающей одновременно значением данных и ключом сортировки. Значение ключа уникально и применяется для автоматической сортировки данных.

## Постановка задачи

В рамках данной курсовой работы необходимо разработать консольное приложение. 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 Map;
* выполнить сравнение времени выполнения перечисленными выше способами реализации следующих операций: создание множества из N элементов, мощность множества, подмножество A⊂A, подмножество 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;
* 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.

## Анализ требований

### Требования к интерфейсу пользователя

Согласно техническому заданию интерфейс приложения представляет собой приложение с удобным вводом исходных данных и наглядным выводом результирующей информации.

Приложение для сравнения структур данных при работе с множествами должно иметь интерфейс, позволяющий:

* задавать мощность создаваемого множества;
* выводить наименование операции, выполненной над множествами;
* выводить тип структуры данных, использованной для выполнения операций;
* выводить время выполнения операций над множествами.

По требуемым возможностям интерфейса является обоснованной реализация интерфейса в форме консольного приложения. Вывод временных характеристик осуществляется в таблицу, заголовки столбцы которой – структуры данных, а заголовки строк – наименования основных операций над ними.

В качестве графической оболочки возможно использование консоли Windows. Консоль Windows – это инфраструктура для консольных приложений в ОС семейства Microsoft Windows. Экземпляр консоли Windows имеет экранный буфер и буфер ввода. Это позволяет консольным приложениям запускаться внутри окна или в аппаратном текстовом режиме (чтобы занимать весь экран).

### Требования к структурам данных

В курсовой работе используются следующие структуры данных: односвязный список, класс «Список», контейнерные классы List, Set, Multiset, Map. Далее приведены краткие описания каждой структуры данных, в частности, их объявления (листинги 1-6) и основные методы (таблицы 1-6).

typedef struct node {

int value;

struct node\* next;

} Node;

Листинг 1 – Объявление односвязного списка

Таблица 1 – Основные операции с односвязным списком

|  |  |
| --- | --- |
| **Название метода** | **Назначение метода** |
| InitEmptySet | Инициализация пустого списка |
| Length | Текущая мощность |
| AddElement | Добавить элемент в список |
| HasElement | Проверяет наличие элемента в списке |
| PrintSet | Вывод списка в строку |
| ClearSet | Освобождение памяти |

class List {

class Node {

int value;

Node\* next;

};

Node\* start;

};

Листинг 2 – Объявление класса «Список»

Таблица 2 – Основные операции с классом «Список»

|  |  |
| --- | --- |
| **Название метода** | **Назначение метода** |
| List | Инициализация пустого списка |
| length | Текущая мощность |
| addElement | Добавить элемент в список |
| hasElement | Проверяет наличие элемента в списке |
| print | Вывод списка в строку |
| ~List | Освобождение памяти |

list<int> container;

Листинг 3 – Объявление контейнера List

Таблица 3 – Основные операции контейнера List

|  |  |
| --- | --- |
| **Название метода** | **Назначение метода** |
| list | Инициализация пустого списка |
| size | Текущая мощность |
| push\_back | Добавить элемент в конец списка |
| find | Возвращает итератор на элемента списка |
| empty | Проверяет наличие элементов |
| clear | Удаление элементов списка |

set<int> container;

Листинг 4 – Объявление контейнера Set

Таблица 4 – Основные операции контейнера Set

|  |  |
| --- | --- |
| **Название метода** | **Назначение метода** |
| set | Инициализация пустого множества |
| size | Текущая мощность |
| insert | Добавить элемент в множество |
| contains | Проверяет наличие элемента в множестве |
| empty | Проверяет наличие элементов |
| clear | Удаление элементов множества |

multiset<int> container;

Листинг 5 – Объявление контейнера Multiset

Таблица 5 – Основные операции контейнера Multiset

|  |  |
| --- | --- |
| **Название метода** | **Назначение метода** |
| multiset | Инициализация пустого множества |
| size | Текущая мощность |
| insert | Добавить элемент в множество |
| contains | Проверяет наличие элемента в множестве |
| empty | Проверяет наличие элементов |
| clear | Удаление элементов множества |

map<int, int> container;

Листинг 6 – Объявление контейнера Map

Таблица 6 – Основные операции контейнера Map

|  |  |
| --- | --- |
| **Название метода** | **Назначение метода** |
| map | Инициализация пустого словаря |
| size | Текущая мощность |
| insert | Добавить элемент в словарь |
| contains | Проверяет наличие ключа в словаре |
| empty | Проверяет наличие элементов |
| clear | Удаление элементов словаря |

Все перечисленные выше структуры данных возможно использовать для реализации «Множества» с функциями F1-F14.

### Требования к программным средствам

Функциональные требования к приложению следующие:

* Ввести мощность множества
* Вычислить время выполнения операций над множествами
* Вывести результат сравнения

В соответствии с функциональными требованиями к курсовой работе была составлена диаграмма вариантов использования, которая отражает возможности приложения.

Ниже представлено краткое описание вариантов использования, отображенных на диаграмме (рисунок 1).

«Вычислить время выполнения операций» – вычисление времени выполнения каждой операции при использовании одной из реализаций «Множества».

Варианты использования начиная с «Создать множество» по «Найти симметричную разность множеств» включаются в вариант использования «Вычислить время выполнения операций» и осуществляют выполнение операций над множествами.

«Вывести результат» – вывод в консоль времени выполнения набора операций над множествами.

«Ввести мощность множества» – ввод мощности множества в консоль.

Варианты использования начиная с «Вывести результаты для односвязного списка» по «Вывести результаты для Map» включаются в вариант использования «Вывести результат» и осуществляют консольный вывод времени выполнения набора операция для конкретной реализации «Множества».

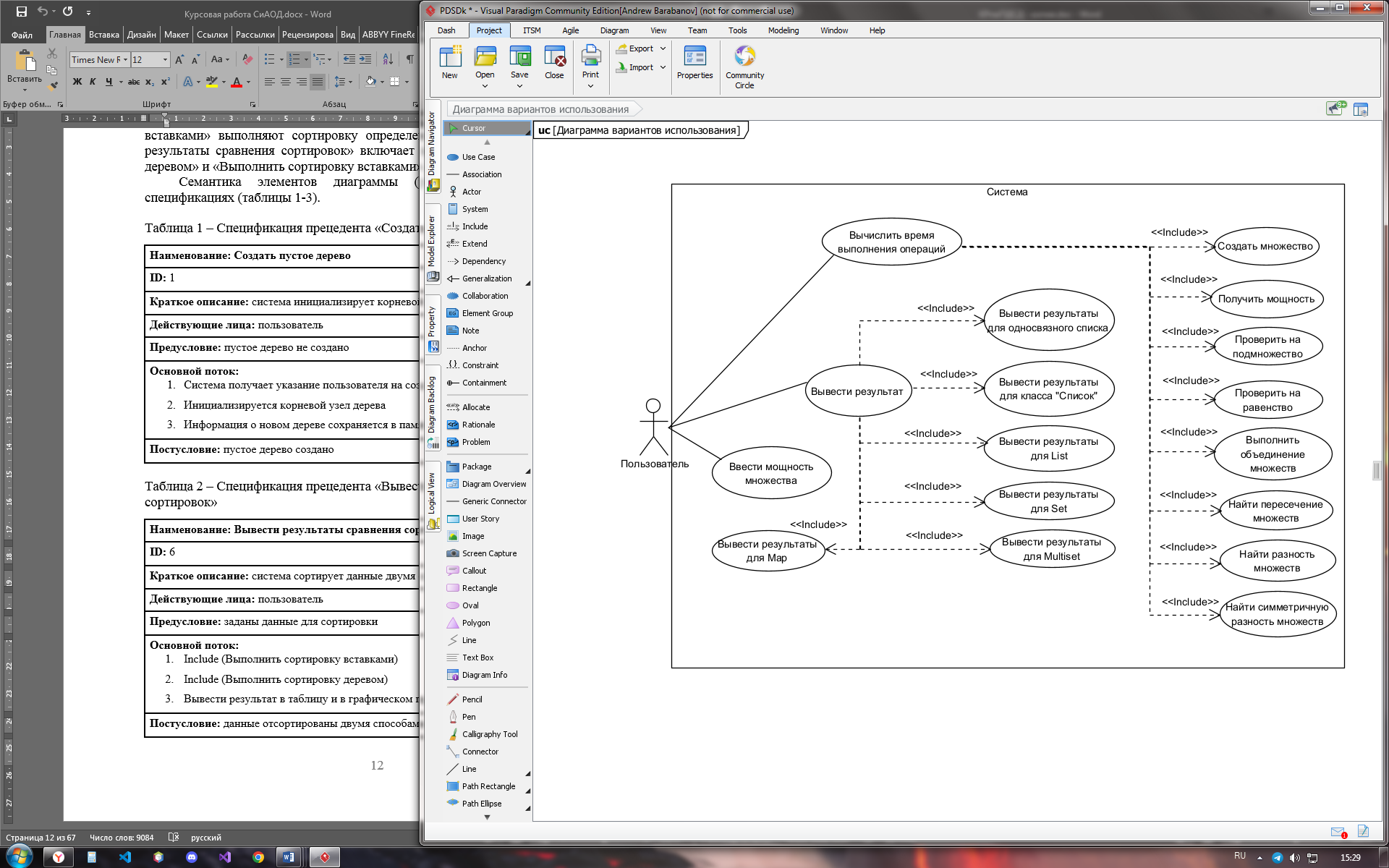


Рисунок 1 – Диаграмма вариантов использования

Семантика элементов диаграммы (рисунок 1) представлена в спецификациях (таблицы 7-9).

Таблица 7 – Спецификация прецедента «Ввести мощность множества»

|  |
| --- |
| **Наименование: Ввести мощность множества** |
| **ID:** 10 |
| **Краткое описание:** система инициализирует корневой узел нового дерева |

Продолжение табл. 7

|  |
| --- |
| **Действующие лица:** пользователь |
| **Предусловие:** мощность множества не задана |
| **Основной поток:**   1. Система запрашивает значение мощности создаваемого множества 2. Значение обрабатывается на корректность 3. Выводится информация о результате обработки ввода |
| **Постусловие:** мощность множества установлена |

Таблица 8 – Спецификация прецедента «Вычислить время выполнения…»

|  |
| --- |
| **Наименование: Вычислить время выполнения операций** |
| **ID:** 1 |
| **Краткое описание:** система вычисляет время, затраченное на выполнение операций |
| **Действующие лица:** пользователь |
| **Предусловие:** заданы мощности множеств A и B |
| **Основной поток:**   1. Include (Создать множество) 2. Include (Получить мощность) 3. Include (Проверить на подмножество) 4. Include (Проверить на равенство) 5. Include (Выполнить объединение множеств) 6. Include (Найти пересечение множеств) 7. Include (Найти разность множеств) 8. Include (Найти симметричную разность множеств) |
| **Постусловие:** подсчитано время выполнения операций |

Таблица 9 – Спецификация прецедента «Вывести результаты для List»

|  |
| --- |
| **Наименование: Вывести результаты для List** |
| **ID:** 13 |
| **Краткое описание:** система выводит в таблицу время выполнения для List |
| **Действующие лица:** система |
| **Предусловие:** вычислено время выполнения операций для List |

Продолжение табл. 9

|  |
| --- |
| **Основной поток:**   1. Система получает из памяти значения времени выполнения операций для List 2. Числовые данные конвертируются в строку 3. Выводится информация о результате с использованием табуляции |
| **Постусловие:** время выполнения для List выведено в таблицу |

Необходимо отметить, что связь включения обоснована постоянством выполнения подфункций при вызове основной функции вне зависимости от действий пользователя.

# Проектирование программы

## Модель интерфейса

В рамках этапа проектирования был разработан пользовательский интерфейс консольного приложения. Скриншот окна консольного приложения для ввода мощности множеств и вывода временных результатов выполнения операций представлен ниже (рисунок 2).

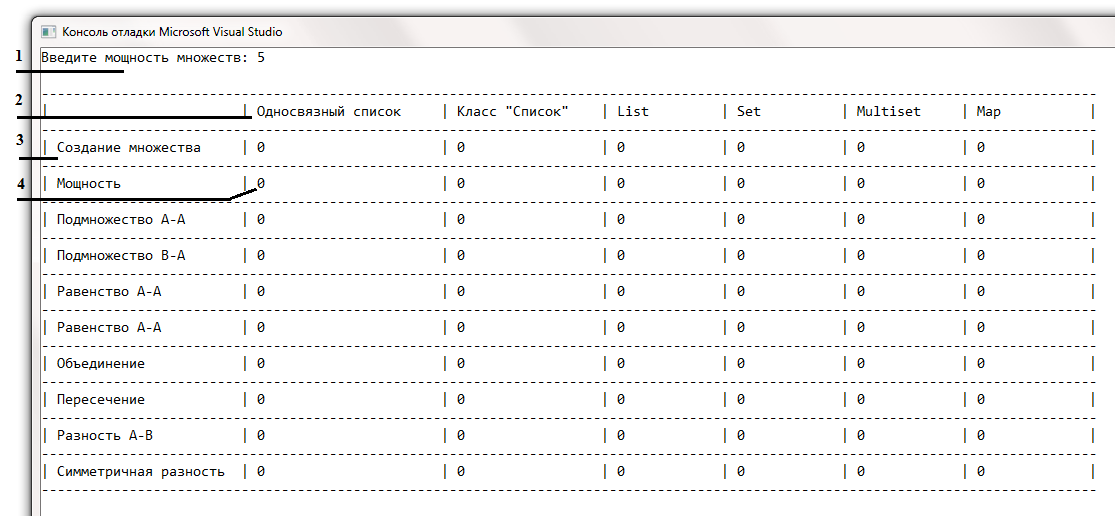


Рисунок 2 – Модель интерфейса

Ключевые элементы пользовательского интерфейса, созданные на основе стандартной системы консольного ввода/вывода с использованием выравнивания:

1. Строка ввода мощности создаваемых множеств (целочисленное неотрицательное значение).
2. Столбцы таблицы – тип структуры данных, используемый для выполнения операции.
3. Строка таблицы – выполняемая операция.
4. Ячейка таблицы представляет целочисленный результат измерения времени выполнения конкретной операции для той или иной структуры данных.

Данный интерфейс отвечает требованиям к курсовой работе, рассмотренным ранее (пункт 1.3.1), а также предоставляет возможность наглядно сравнить результаты измерений.

## Проектирование структур данных

В курсовой работе используются следующие структуры данных: односвязный список, класс «Список», контейнерные классы List, Set, Multiset, Map.

Ниже приведено описание СД – «Односвязный список» и функции «Подмножество» (листинги 7-8).

typedef struct straightlist {

int value;

struct straightlist\* next;

} Node;

Листинг 7 – Описание СД «Односвязный список»

bool IsSub(Node\* a, Node\* b) {

if (IsEmpty(a))

return true;

if (Length(a) > Length(b))

return false;

do {

if (!HasElement(b, a->value))

return false;

a = a->next;

} while (a);

return true;

}

Листинг 8 – Код функции «Подмножество»

Структура straightlist является узлом односвязного списка, который содержит поле значения (value) и указатель на следующий узел (next).

Функция IsSub() проверяет: является множество a подмножеством b, возвращает соответствующие логическое значение.

Node\* a – указатель на голову первого множества.

Node\* b – указатель на голову второго множества.

Здесь используются функции IsEmpty() и Length() для получения информации о мощности множеств, а также функция HasElement() для проверки наличия определенного значения во множестве при его проходе в цикле do-while.

Ниже приведено описание СД – «Класс «Список» и функции «Равенство» (листинги 9-10).

class List {

struct Node {

int value;

Node\* next;

};

Node\* head;

};

Листинг 9 – Описание СД «Класс «Список»

bool List::isEqual(List& b) {

return this->isSubset(b) && b.isSubset(\*this);

}

Листинг 10 – Код функции «Равенство»

Класс List является реализацией множества на основе односвязного списка. Поле head хранит указатель на голову списка. Вложенная структура Node является узлом односвязного списка, который содержит поле значения (value) и указатель на следующий узел (next).

Функция isEqual() проверяет: является текущее множество this равным множеству b, возвращает соответствующие логическое значение.

List& b – ссылка на экземпляр второго множества.

Здесь используется функция isSubset() для использования того факта, что множества равны между собой, если являются подмножествами друг друг друга.

Ниже приведено описание СД – «Контейнер List» и функции «Объединение» (листинги 11-12).

class SetList {

list<int> container;

};

Листинг 11 – Описание СД «Контейнер List»

SetList\* SetList::unite(SetList& a, SetList& b) {

SetList\* c = new SetList();

if (!a.isEmpty()) {

list<int>::iterator iter = a.begin();

do c->add(\*(iter++));

while (iter != a.end());

}

if (!b.isEmpty()) {

list<int>::iterator iter = b.begin();

do c->add(\*(iter++));

while (iter != b.end());

}

return c;

}

Листинг 12 – Код функции «Объединение»

Класс SetList является реализацией множества на основе контейнера list. Поле container хранит экземпляр контейнера.

Функция unite() формирует объединение двух множеств и возвращает указатель на новое множество.

SetList& a – ссылка на экземпляр первого множества.

SetList& b – ссылка на экземпляр второго множества.

Здесь используется функция isEmpty() для получения информации о мощности множеств, а также функция add() для формирования объединения. Обход контейнера происходит в циклах do-while с использованием итераторов контейнера.

Ниже приведено описание СД – «Контейнер Set» и функции «Пересечение» (листинги 13-14).

class Set {

set<int> container;

};

Листинг 13 – Описание СД «Контейнер Set»

Set\* Set::cross(Set& a, Set& b) {

if (a.isEmpty() || b.isEmpty())

return nullptr;

Set\* c = new Set();

set<int>::iterator iter = a.begin();

do {

if (b.has(\*iter))

c->add(\*iter);

iter++;

} while (iter != a.end());

return c;

}

Листинг 14 – Код функции «Пересечение»

Класс Set является реализацией множества на основе контейнера set. Поле container хранит экземпляр контейнера.

Функция cross() формирует пересечение двух множеств и возвращает указатель на новое множество.

Set& a – ссылка на экземпляр первого множества.

Set& b – ссылка на экземпляр второго множества.

Здесь используется функция isEmpty() для получения информации о мощности множеств, функция has() для проверки наличия определенного значения во множестве, а также функция add() для формирования объединения. Обход контейнера происходит в циклах do-while с использованием итераторов контейнера.

Ниже приведено описание СД – «Контейнер Map» и функции «Разность» (листинги 15-16).

class SetMap {

map<int, int\*> container;

};

Листинг 15 – Описание СД «Контейнер Map»

SetMap\* SetMap::substraction(SetMap& a, SetMap& b) {

if (a.isEmpty())

return nullptr;

if (b.isEmpty())

return new SetMap(a);

SetMap\* c = new SetMap();

map<int, int\*>::iterator iter = a.begin();

do {

if (!b.has(iter->first))

c->add(iter->first);

iter++;

} while (iter != a.end());

return c;

}

Листинг 16 – Код функции «Разность»

Класс SetMap является реализацией множества на основе контейнера map. Поле container хранит экземпляр контейнера. Примеч.: в словаре ключ рассматривается как значение множества, второе значение не используется и хранит пустой указатель.

Функция substraction() формирует разность двух множеств и возвращает указатель на новое множество.

SetMap& a – ссылка на экземпляр первого множества.

SetMap& b – ссылка на экземпляр второго множества.

Здесь используется функция isEmpty() для получения информации о мощности множеств, функция has() для проверки наличия определенного значения во множестве, а также функция add() для формирования объединения. Обход контейнера происходит в циклах do-while с использованием итераторов контейнера.

Ниже приведено описание СД – «Контейнер Multiset» и функции «Симметричная разность» (листинги 17-18).

class SetMultiset {

multiset<int> container;

};

Листинг 17 – Описание СД «Multiset»

SetMultiset\* simmetricalSub(SetMultiset& a, SetMultiset& b) {

SetMultiset\* \_unite = unite(a, b);

SetMultiset\* \_cross = cross(a, b);

SetMultiset\* \_sub = substraction(\*\_unite, \*\_cross);

delete \_unite;

delete \_cross;

return \_sub;

}

Листинг 18 – Код функции «Симметричная разность»

Класс SetMultiset является реализацией множества на основе контейнера multiset. Поле container хранит экземпляр контейнера.

Функция simmetricalSub() формирует симметричную разность двух множеств и возвращает указатель на новое множество.

SetMultiset& a – ссылка на экземпляр первого множества.

SetMultiset& b – ссылка на экземпляр второго множества.

Здесь используются функции unite(), cross(), substraction() для последовательного формирования результирующего множества.

## Структура программного обеспечения

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

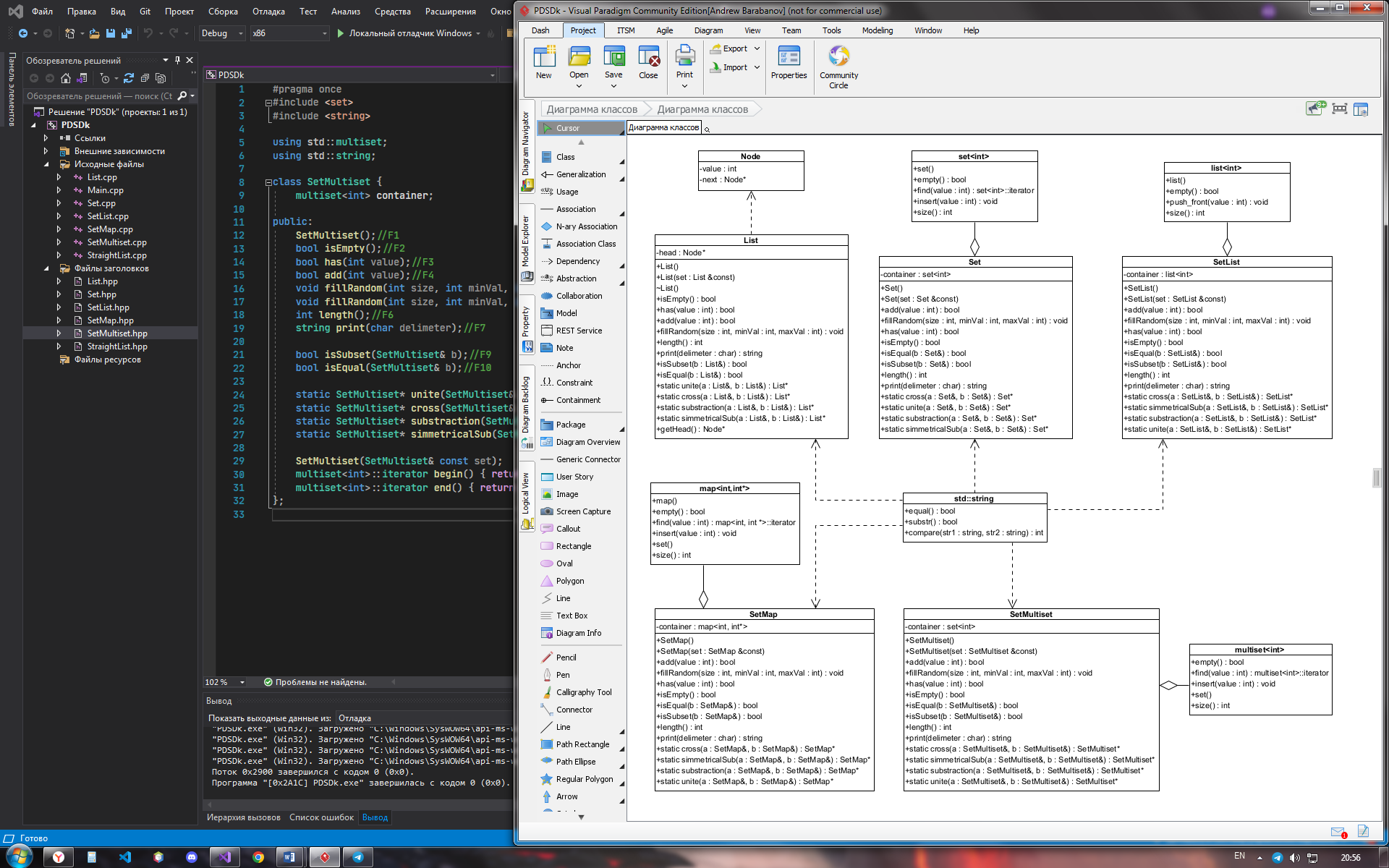


Рисунок 3 – Диаграмма классов

В классах присутствуют общие методы, выполняющие одинаковые действия при работе с множествами, поэтому имеет смысл описать их обособленно на примере класса Set.

add(int value) – добавляет элемент value в множество. Возвращает true, если элемент добавлен; иначе false.

fillRandom(int size, int minVal, int maxVal) – создает множество из size случайных элементов, удовлетворяющих заданным границам значений.

has(int value) – проверяет наличие элемента в множестве. Возвращает true, если элемент присутствует; иначе false.

isEmpty() – проверяет множество на наличие элементов. Возвращает true, если множество пустое; иначе false.

isSubset(Set& b) – выполняет операцию «Проверить на подмножество». Доступ к множеству b осуществляется по ссылке. Возвращает true, если this является подмножеством b; иначе false.

isEqual(Set& b) – выполняет операцию «Проверить на равенство». Доступ к множеству b осуществляется по ссылке. Возвращает true, если this является подмножеством b; иначе false.

length() – выполняет операцию «Получить мощность».

print(char delimeter) – формирует строку, содержащую элементы множества, delimeter является разделителем.

Следующие статические методы классов определяют бинарные операции с созданием нового множества.

static cross(Set& a, Set& b) – выполняет операцию «Найти пересечение множеств».

static unite(Set& a, Set& b) – реализует операцию «Выполнить объединение множеств».

static substraction(Set& a, Set& b) – выполняет операцию «Найти разность множеств».

static simmetricalSub(Set& a, Set& b) – выполняет операцию «Найти симметричную разность множеств».

Ниже приведены спецификации классов.

Класс std::string инкапсулирует работу со строками, предоставляет удобный функционал. Является часть стандартной библиотеки.

equal() – проверяет равенство строк.

substr() – выделяет подстроку.

compare(string str1, string str 2) – сравнивает две строки.

Класс List является реализацией структуры «Множество» на основе односвязного списка (struct Node). List связан с классом Node зависимостью, т.к. первый использует для своей работы указатели Node\*.

Node\* head – указатель на голову списка.

List() – конструктор по умолчанию, создающий пустое множество.

List(List& const set) – конструктор копирования (создает внутри нового объекта класса копию односвязного списка).

~List() – деструктор, очищающий динамически выделенную память при работе с односвязным списком.

getHead() – геттер для головы списка.

Структура Node является реализацией узла односвязного списка.

int value – значение узла.

Node\* next – указатель на следующий узел.

Класс Set является реализацией структуры «Множество» на основе контейнера set (set<int>). Set связан с классом set<int> агрегацией, т.к. первый содержит внутри себя контейнер set<int>.

set<int> container – экземпляр контейнера.

Set() – конструктор по умолчанию, создающий пустое множество.

Set(Set& const set) – конструктор копирования (создает внутри нового объекта класса копию контейнера).

Класс set<int> является контейнерным классом множества в стандартной библиотеке.

set() – конструктор пустого контейнера.

empty() – проверяет контейнер на наличие элементов.

find(int value) – возвращает итератор найденного элемента.

insert(int value) – вставляет новый элемент в контейнер.

size() – возвращает кол-во элементов в контейнере.

Класс SetList является реализацией структуры «Множество» на основе контейнера list (list<int>). Set связан с классом list<int> агрегацией, т.к. первый содержит внутри себя контейнер list<int>.

list<int> container – экземпляр контейнера.

SetList() – конструктор по умолчанию, создающий пустое множество.

SetList(SetList& const set) – конструктор копирования (создает внутри нового объекта класса копию контейнера).

Класс list<int> является контейнерным классом списка в стандартной библиотеке.

list() – конструктор пустого контейнера.

empty() – проверяет контейнер на наличие элементов.

push\_front() – добавляет элемент в начало списка.

size() – возвращает кол-во элементов в контейнере.

Класс SetMap является реализацией структуры «Множество» на основе контейнера map (map<int, int\*>). Set связан с классом map<int, int\*> агрегацией, т.к. первый содержит внутри себя контейнер map<int, int\*>.

map<int, int\*> container – экземпляр контейнера.

SetMap() – конструктор по умолчанию, создающий пустое множество.

SetMap(SetMap& const set) – конструктор копирования (создает внутри нового объекта класса копию контейнера).

Класс map<int, int\*> является контейнерным классом словаря в стандартной библиотеке.

map() – конструктор пустого контейнера.

empty() – проверяет контейнер на наличие элементов.

find(int value) – возвращает итератор найденного элемента.

insert(int value) – вставляет новый элемент в контейнер.

size() – возвращает кол-во элементов в контейнере.

Класс SetMultiset является реализацией структуры «Множество» на основе контейнера multiset (multiset<int>). Set связан с классом multiset<int> агрегацией, т.к. первый содержит внутри себя контейнер multiset<int>.

multiset<int> container – экземпляр контейнера.

SetMultiset() – конструктор по умолчанию, создающий пустое множество.

SetMultiset(SetMultiset& const set) – конструктор копирования (создает внутри нового объекта класса копию контейнера).

Класс multiset<int> является контейнерным классом множества в стандартной библиотеке.

multiset() – конструктор пустого контейнера.

empty() – проверяет контейнер на наличие элементов.

find(int value) – возвращает итератор найденного элемента.

insert(int value) – вставляет новый элемент в контейнер.

size() – возвращает кол-во элементов в контейнере.

# Реализация программы

## Кодирование

В ходе выполнения курсовой работы было разработано приложение «Сравнение контейнеров при реализации СД «Множество», код которого приведен в приложении А.

Для алгоритма операции «Разность множеств» (класс «Список») была разработана диаграмма деятельности (рисунок 4).

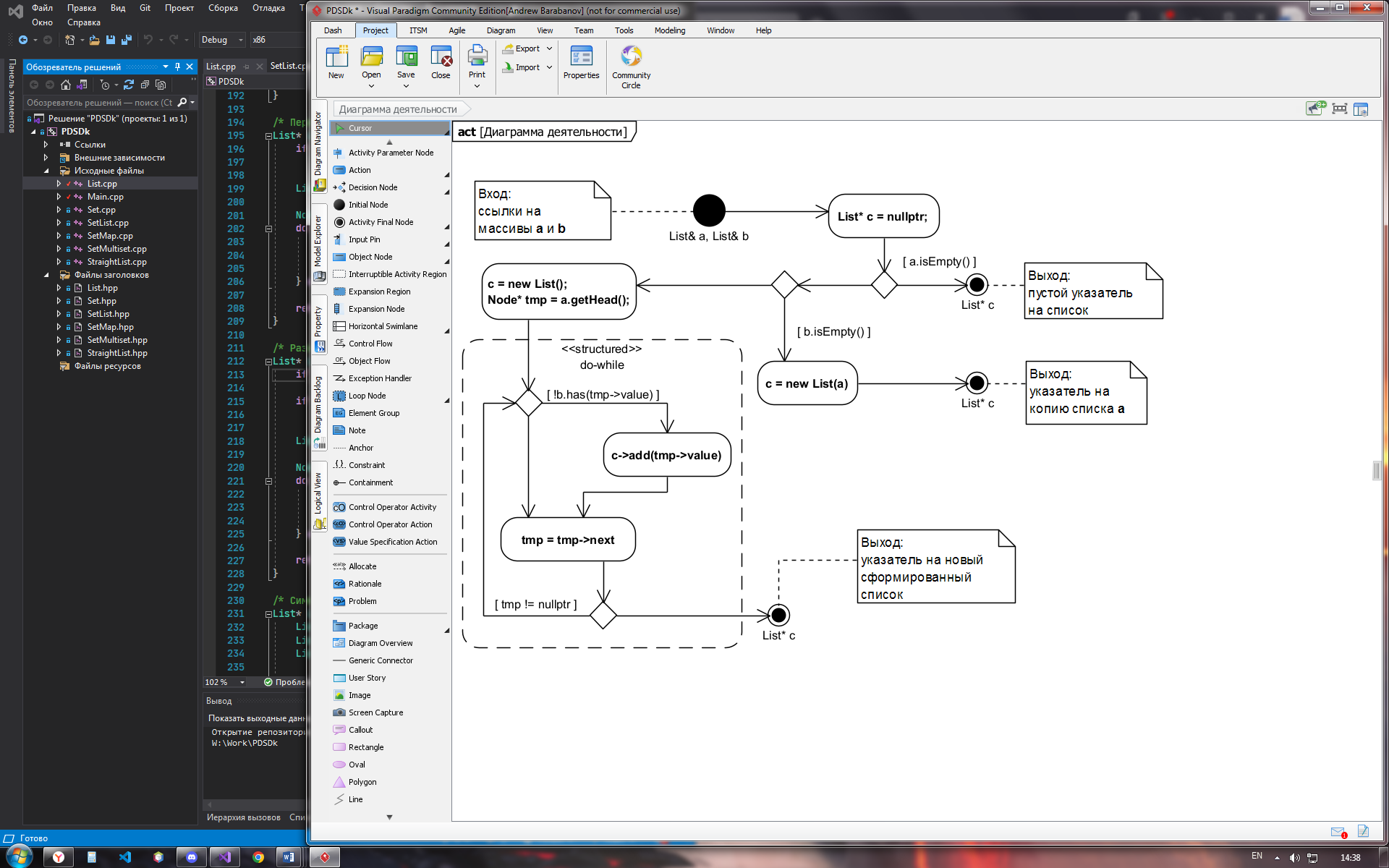


Рисунок 4 – Диаграмма деятельности

На диаграмме используются следующие переменные:

* a – ссылка на первый список;
* b – ссылка на второй список;
* с – указатель на список-результат;
* tmp – рабочий указатель на текущий узел списка.

Ниже приведен основной пояснительный текст к диаграмме деятельности (рисунок 4).

На вход функции подаются две ссылки на объекты класса List (реализация структуры «Множество» с использованием принципов ООП), затем последовательно обрабатывается наличие элементов во множестве **a** (если isEmpty(), то выход с пустым указателем) и множестве **b** (если isEmpty(), то выход с созданием копии **a**).

Далее программа динамически создает новое множество **с**, которое будет являться результатом выполнения операции «Разность множеств», и создает рабочий указатель на голову списка **a**.

В последующем цикле do-while происходит последовательное формирование множества **c** из элементов множества **a**, которые не содержатся в **b** (внутренне условие [!b.has(tmp->value)]). На каждой итерации цикла в tmp заносится указатель на следующий узел списка. Цикл выполняется до тех пор, пока указатель на следующий элемент не будет пустым (nullptr).

Выполнение функции после выхода из цикла заканчивается возвращением указателя на сформированное множество **c**.

## 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).

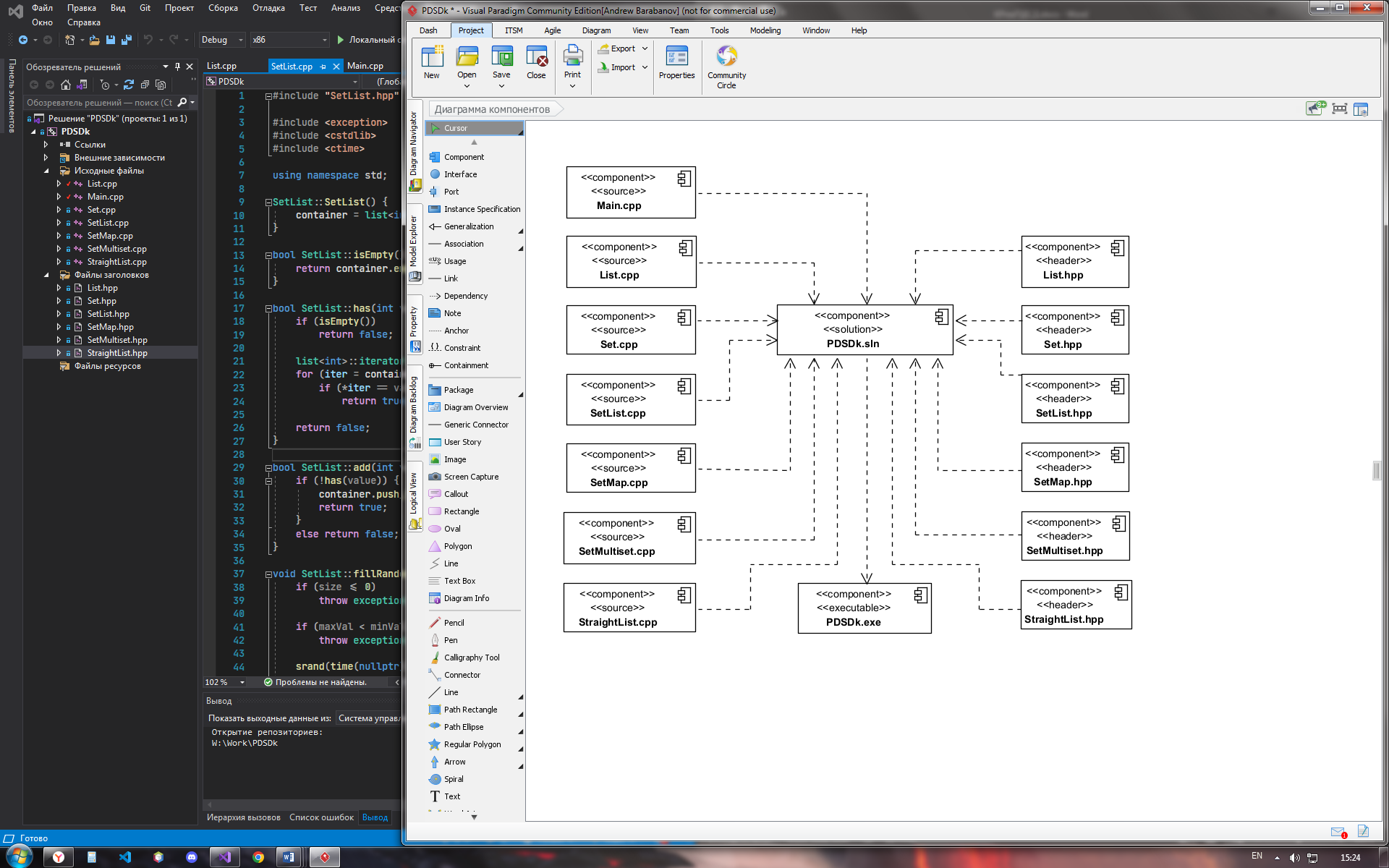


Рисунок 5 – Диаграмма компонентов

Реализация разделена на семь файлов:

1. Main.cpp – файл точки входа консольного приложения main().
2. List.cpp – файл реализации «Множества» классом «Список».
3. Set.cpp – файл реализации «Множества» с помощью контейнера set.
4. SetList.cpp – файл реализации «Множества» с помощью контейнера list.
5. SetMap.cpp – файл реализации «Множества» с помощью контейнера map.
6. SetMultiset.cpp – файл реализации «Множества» с помощью контейнера multiset.
7. StraightList.cpp – файл реализации «Множества» на основе односвязного списка.

Связь модулей программ обеспечивается за счет шести заголовочных файлов:

1. List.hpp – заголовочный файл класса «Список».
2. Set.hpp – заголовочный файл контейнера set.
3. SetList.hpp – заголовочный файл контейнера list.
4. SetMap.hpp – заголовочный файл контейнера map.
5. SetMultiset.hpp – заголовочный файл контейнера multiset.
6. StraightList.hpp – заголовочный файл односвязного списка.

Остальные артефакты:

PDSDk.exe – исполняемый файл проекта.

PDSDk.sln – файл решения проекта.