**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 unordered\_map и unordered\_multimap»

ПГУ 09.03.04 - 03КР212.16 ПЗ

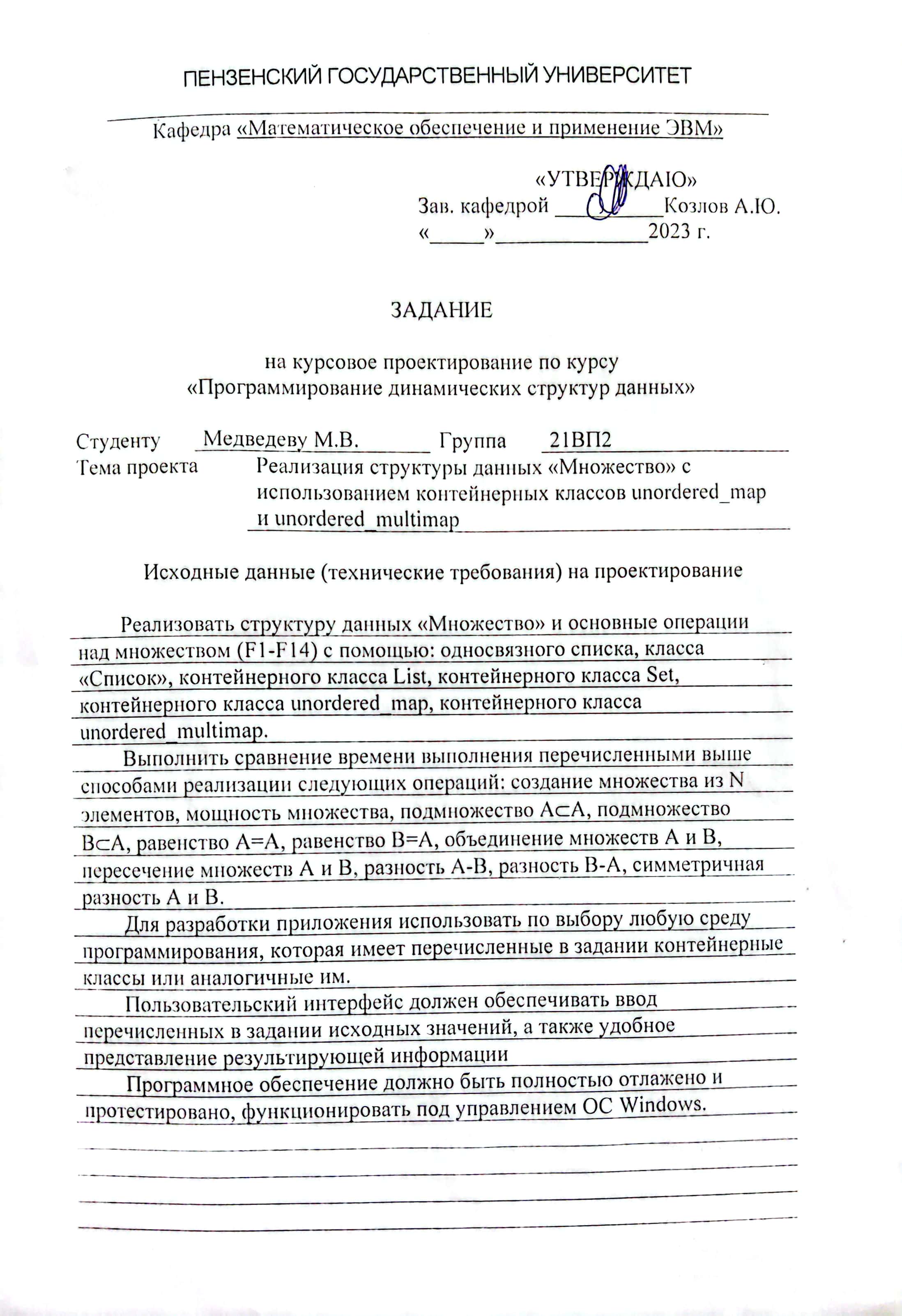
Направление подготовки – 09.03.04 Программная инженерия

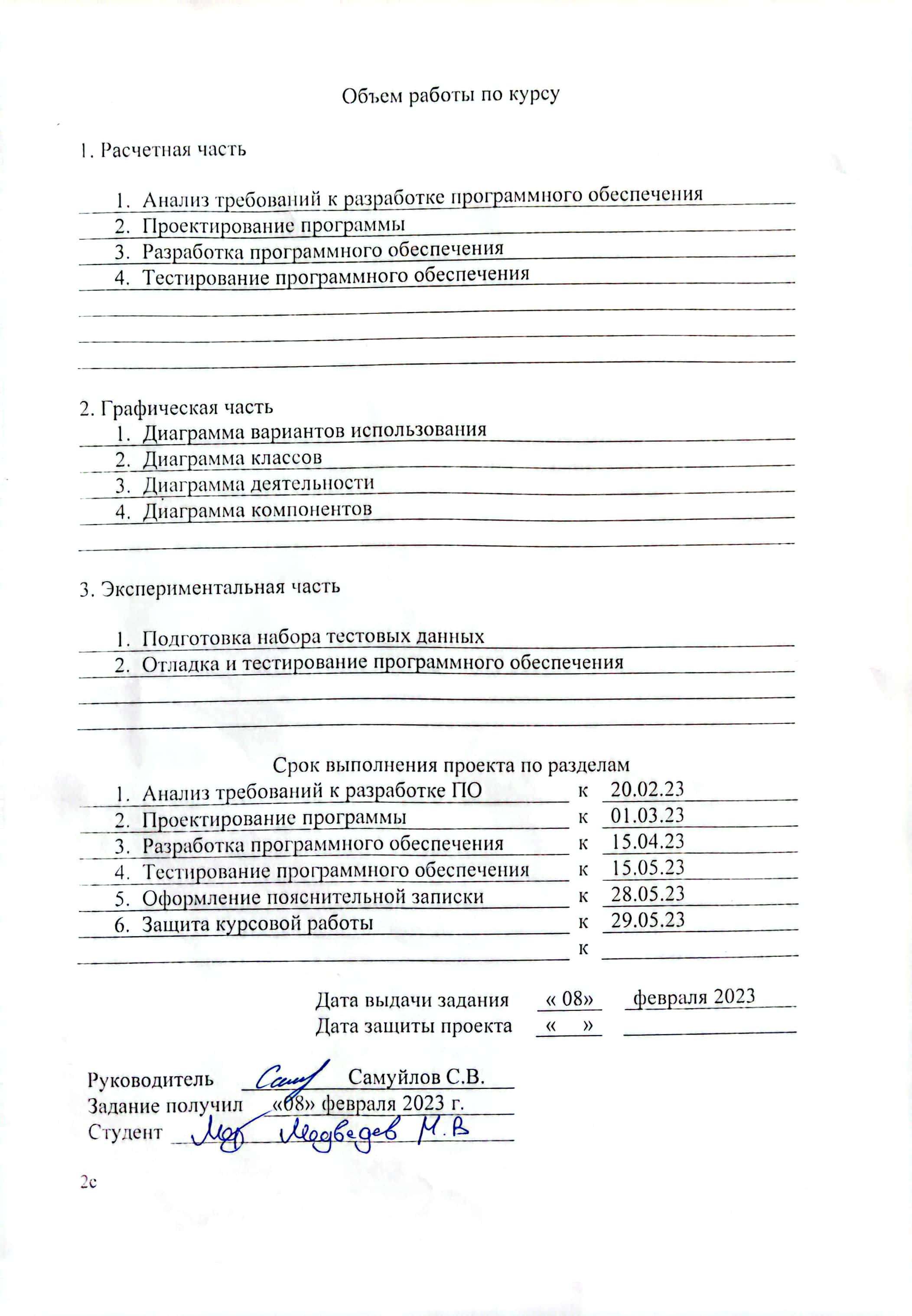
Профиль подготовки – Программная инженерия

|  |  |  |
| --- | --- | --- |
| Выполнил студент |  | Медведев М.В. |
| Группа |  | 21ВП2 |
|  |  |  |
| Руководитель |  |  |
| к.т.н., доцент |  | Самуйлов С.В. |

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

2023





# 1. Постановка задачи и анализ предметной области

## 1.1 Основные понятия и определения

Множество – структура данных, представляющая из себя набор неповторяющихся данных одного и того же типа. Оно отличается от всех других структур данных, прежде всего тем, что все операции происходят в теоретико-множественном смысле.

Множество состоит из элементов. Если количество элементов множества известно или счётно, множество – конечно, в противном случае – бесконечно. При добавление уже имеющийся в множестве элемента, множество не меняется, а при удалении отсутствующего в множестве элемента не возникает ошибочной ситуации.

Мощность множества определяется количеством содержащихся в нём элементов. Равномощными называют множества с одинаковой мощностью. Пустое множество не содержит элементов, его мощность равна 0, обозначается ∅. Множество A является подмножеством множества B, если все элементы A являются элементами B, то есть входят в состав множества B. Записывается следующим образом: A⊂B. Множество A равно множеству B, если множество A – подмножество B, а множество B – подмножество A (A⊆B – взаимное включение): все элементы множества A равны элементам множества B и наоборот. Мощности таких множеств, естественно, не отличаются. Объединение множеств A и B (A∪B) представляет собой новое множество C, состоящее из элементов множества A и элементов множества B (в случае наличия совпадающих (равных) элементов в множествах A, B добавление элемента происходит только 1 раз, иначе C – мультимножество, а не множество). Если отобрать одинаковые элементы в множествах A и B и создать из них новое множество C, можно получить пересечение множеств A и B (A∩B). При отсутствии таких элементов C будет пустым множеством. Разностью множеств А и В (A\B) называется множество, содержащее все элементы, которые принадлежат множеству А и не принадлежат множеству В. Симметрическая разность двух множеств – это множество, включающее все элементы исходных множеств, не принадлежащие одновременно обоим исходным множествам. Это множество будет состоять из элементов множества A, не входящих в B, и множества B, не входящих в A.

Контейнерным называется класс, который предназначен для хранения каким-либо образом организованных данных и работы с ними. Контейнерные классы подразделяются на последовательные и ассоциативные. В последовательных элементы расположены в памяти непрерывно (последовательно, друг за другом), в ассоциативных иначе: они автоматически сортируются согласно определённому принципу, обеспечивают быстрый доступ к данным по ключу.

В курсовой работе будут рассмотрены следующие контейнерные классы: List, Set, unordered\_map и unordered\_multimap.

Контейнерный класс List представляет собой двусвязный список, где каждый элемент имеет указатели на предыдущий и следующий элементы. Это последовательный контейнер, в котором элементы расположены непрерывно. Он предоставляет эффективный способ добавления, удаления и перемещения элементов в списке, в то время как доступ к произвольному элементу происходит за линейное время.

Контейнерный класс Set представляет собой отсортированное множество уникальных элементов. Так как Set – ассоциативный контейнер, элементы располагаются не последовательно, а по принципу бинарного дерева. Благодаря этому множество автоматически сортируется по возрастанию и не может иметь дубликатов. Это особенно удобно для хранения данных, когда важен только факт присутствия элемента, но не его порядок.

Контейнерный класс unordered\_map представляет из себя одну из реализаций ассоциативного контейнера Map. Map практически такой же контейнер как и Set, с такими же принципами построения. Отличие лишь в том, что элемент в Map представляет из себя пару ключ: значение, бинарное дерево строится на основе ключей, а значения уже непосредственно с ними связываются. Map хранит уникальные ключи. Отличие unordered\_map от map заключается в том, что вместо бинарных деревьев данный контейнер представляет из себя хеш-таблицу, а порядок сортировки задаётся хеш-функцией. За счёт этого появляется мгновенный доступ к ключу и другие преимущества хеш-таблиц.

Контейнерный класс unordered\_multimap это такой же контейнер, как и unordered\_map, но он, в свою очередь, позволяет хранить неуникальные элементы.

## 1.2 Постановка задачи на разработку программы

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 unordered\_map, контейнерного класса unordered\_multimap.

Выполнить сравнение времени выполнения перечисленных 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 множества A и B, разность A-B, разность B-A, симметричная разность A и B.

Для разработки приложения будет использована среда программирования Visual Studio 2022 для языка C++, содержащего перечисленные контейнерные классы.

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.

## 1.3 Анализ требований

### 1.3.1 Требования к интерфейсу пользователя

Разрабатываемое приложение должно предоставлять пользователю удобный интерфейс, позволяющий вводить мощность каждого из множеств. Интерфейс необходим для того, чтобы пользователь мог проанализировать результаты работы программы и удостовериться в правильном её исполнении.

Задача курсовой работы – сравнение времени выполнения операций над «Множеством», как структурой данных, реализованной различными способами, следовательно наиболее оптимальным будет представление результата в виде таблицы, заголовки столбцы которой – структуры данных, а заголовки строк – наименования основных операций над ними. Интерфейс будет представлен в виде консольного приложения.

### 1.3.2 Требования к структуре данных

В данной курсовой работе используются следующие структуры данных:

* Односвязный список
* Класс «Список»
* Контейнерный класс «List»
* Контейнерный класс «Set»
* Контейнерный класс «unordered\_map»
* Контейнерный класс «unordered\_multimap»

Пример объявления **односвязного списка**:

Set\* A = createSet(7, 10, 99);

Set\* B = createEmptySet();

Ячейка односвязного списка состоит из двух частей: элемент и адрес следующего элемента в памяти. Благодаря этому образуется цепочка связанных друг с другом ячеек, с помощью которой можно впоследствии взаимодействовать со списком и элементами, которые он хранит. Чтобы использовать список в функциях, необходимо передавать в них указатель на начало списка Set\* start.

Основные операции представлены в таблице 1.

Таблица 1 – основные операции односвязного списка

|  |  |
| --- | --- |
| **Операция** | **Описание операции** |
| Set\* createEmptySet() | Создает пустое множество |
| bool isSetEmpty(Set\* start) | Проверка на пустое множество |
| bool isElemInSet(Set\* start, int element) | Проверяет принадлежность элемента множеству |
| Set\* addElemInSet(Set\* start, int element) | Добавляет новый элемент в множество |
| Set\* createSet(int elemCount, int rangeDown, int rangeUp) | Создает множество |
| int setPower(Set\* start) | Определяет мощность множества |
| string printSet(Set\*start, string separator) | Выводит элементы множества |
| Set\* deleteSet(Set\* start) | Удаление множества |
| bool isSubSet(Set\* set, Set\* subset) | Проверка на подмножество А-В |
| bool setsEquals(Set\* first, Set\* second) | Равенство двух множеств А-В. |
| Set\* setsUnification(Set\* first, Set\* second) | Объединение двух множеств. |
| Set\* setsIntersection(Set\* first, Set\* second) | Пересечение двух множеств. |
| Set\* setsDifference(Set\* first, Set\* second) | Разность множеств. |
| Set\* setsSymmetricDifference(Set\* first, Set\* second) | Симметричная разность. |

Пример объявления **множества на основе класса «Список»**:

SetList A;

A.create(9, 10, 99);

Класс «Список» по своей структуре схож с односвязным списком, однако в его реализации присутствует парадигма ООП и вместо функций используются методы. При вызове конструктора класса Set инициализируется List\* head = NULL, который является указателем на первый элемент списка.

Основные методы класса «Список» представлены в таблице 2.

Таблица 2 – основные методы класса «Список»

|  |  |
| --- | --- |
| **Метод** | **Описание метода** |
| bool empty() | Проверка на пустое множество |
| bool find(int element) | Проверяет принадлежности элемента множеству |
| void push\_front(int element) | Добавляет нового элемента в множество |
| int power() | Определяет мощность множества |
| string print(string separator) | Выводит элементы множества с символьным разделителем |
| bool isSubset(SetList\* someSet) | Проверка на подмножество |
| bool setEquals(SetList\* someSet) | Равенство двух множеств |
| SetList\* setUnification(SetList\* someSet) | Объединение двух множеств. |
| SetList\* setIntersection(SetList\* someSet) | Пересечение двух множеств. |
| SetList\* setDifference(SetList\* set) | Разность множеств. |

Пример объявления **контейнерного класса «List»**:

list<int> myList;

Операции над множеством реализуются с помощью методов контейнера «List», которые представлены в таблице 3.

Таблица 3 – основные методы контейнерного класса «List»

|  |  |
| --- | --- |
| **Метод** | **Описание метода** |
| push\_back(value) | Добавить элемент в конец списка |
| push\_front(value) | Добавить элемент в начало списка |
| pop\_back() | Удалить последний элемент списка |
| pop\_front() | Удалить первый элемент списка |
| size() | Получить количество элементов в списке |
| empty() | Проверка на пустой список |
| begin(), end() | Итераторы на начало и конец списка |
| insert(iterator, value) | Вставить элемент перед указанным итератором |
| erase(iterator) | Удалить элемент, на который указывает итератор |
| clear() | Удалить все элементы из списка |

Пример объявления **контейнерного класса «Set»:**

set<int> mySet;

Реализация множества по аналогии с контейнером «List», но вместо него используется «Set». Основные методы контейнера «Set» представлены в таблице 4.

Таблица 4 – основные методы контейнерного класса «Set»

|  |  |
| --- | --- |
| **Метод** | **Описание метода** |
| insert(value) | Добавить элемент в множество |
| erase(value) | Удалить элемент из множества |
| size() | Получить количество элементов в множестве |
| empty() | Проверить, является ли множество пустым |
| begin(), end() | Итераторы на начало и конец множества |
| find(value) | Найти элемент в множестве (возвращает итератор на найденный элемент или end(), если элемент не найден) |
| clear() | Удалить все элементы из множества |

Пример объявления **контейнерного класса «unordered\_map»:**

unordered\_map<int> myUnorderedMap;

Основные методы контейнерного класса «unordered\_map» представлены в таблице 5.

Таблица 5 – основные методы контейнерного класса «unordered\_map»

|  |  |
| --- | --- |
| **Метод** | **Описание метода** |
| at(key) | Получить доступ к элементу по ключу |
| emplace(key, value) | Добавляет элемент, созданный на месте |
| empty(key) | Проверяет отсутствие ключа |
| begin(), end() | Итераторы на начало и конец Map |
| size() | Получить количество элементов |
| clear() | Удалить все элементы |
| find(key) | Найти элемент по ключу |
| bucket\_count() | Количество зарезервированных корзин в хеш-таблице |
| bucket\_size(num) | Размер одной из корзин |
| erase(key) | Удалить элемент по ключу |

Пример объявления **контейнерного класса «unordered\_multimap»:**

unordered\_multimap<int> myUnorderedMultiMap;

Основные методы контейнерного класса «unordered\_multimap» такие же, как и у контейнерного класса «unordered\_map».

### 1.3.3 Требования к программным средствам

Для создания программного обеспечения необходимо сформулировать и проанализировать некоторые требования к курсовой работе. Программа должна полностью покрывать условия технического задания, при этом требуется продумать некоторые детали, исходящие из алгоритма, выполняющего поставленную задачу. Таким образом, правильная формулировка требований и их подробный анализ – это главная цель разработчика перед проектированием и созданием программы.

В техническом задании на курсовой проект определены следующие функциональные требования. Программа должна позволять пользователю вводить мощность множеств, а именно единую размерность односвязного списка, класса «Список», контейнерного класса «List», контейнерного класса «Set», контейнерного класса «unordered\_map», контейнерного класса «unordered\_multimap». Вычислить время операций всех выше раннее перечисленных структур данных. Для сравнения времени выполнения операций, выводить результаты в табличной форме, где заголовки столбцов являются наименованием структур данных, а заголовки строк – наименование операций.

На рисунке 1, приведенном ниже, представлена диаграмма вариантов использования, которая создана с учётом требований из технического задания.

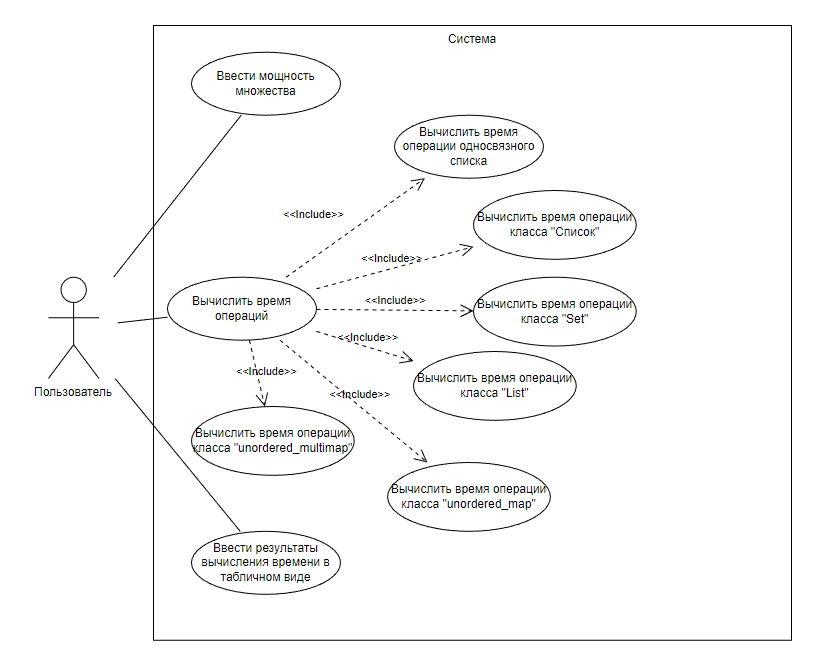


Рисунок 1 – Диаграмма вариантов использования

Сущность, использующая систему, называется актером. Им является пользователь. Функции программы представлены в виде вариантов использования, находящихся внутри системы. Например, на диаграмме варианты «Вычислить время операций односвязного списка», «Вычислить время операций класса List», «Вычислить время операций класса Set» включены в вариант «Вычислить время операций» (отношение «include»). Данные включения обоснованы техническим заданием, поскольку необходимо с помощью вычисления времени операций сравнить работоспособность различных структур данных.

Сценарии вариантов использования «Выбрать тип массива», «Заполнить и отсортировать выбранный массив», а также «Отобразить данные в табличной и графической форме» приведены в таблицах 1 – 3.

Таблица 6 – Сценарий варианта использования «Задать размер множества»

|  |
| --- |
| **Наименование**: ввести мощность множества |
| **ID**: 1 |
| **Краткое описание**: задается размер N для последующего создания множества, также определяется будущая верхняя граница допустимых значений равная 10\*N |
| **Действующие лица**: пользователь |
| **Предусловие**: - |
| **Основной** **поток**:   1. Пользователь вводит размер множества N 2. Система сохраняет введенное значение |
| **Постусловие**: задана размерность будущего множества, введенная пользователем |

Таблица 7 – Сценарий варианта использования «Вычислить время операций контейнерного класса «List»

|  |
| --- |
| **Наименование:** вычислить время операций контейнерного класса «List» |
| ID: 5 |
| **Краткое описание:** система вычисляет время каждой из операций над множеством, созданным на основе контейнерного класса «List» |
| **Действующие лица:** пользователь |
| **Предусловие: -** |
| **Основной поток:**   1. В систему передается размерность контейнерного класса «List», которую указал пользователь 2. Система формирует объект контейнерного класса «List» 3. Определяется начальное и конечное время выполнения каждой операции над множеством 4. Система вычисляет каждое время, затраченное на выполнения той или иной операций над множеством, а также сохраняет времена в отдельные переменные |
| **Постусловие:** вычислено время выполнения каждой операций над множеством, сделанным на основе контейнера «List» |

# 2. Проектирование программы

## 2.1. Модель интерфейса

В процессе разработки ПО было разработано консольное приложение, выполняющее функцию интерфейса приложения. Консольное приложение, сделанное в виде таблицы, полностью покроет все необходимые требования к пользовательскому интерфейсу. Пользовательский интерфейс – средство, обеспечивающее передачу информации между пользователем (человеком) и программно-аппаратными компонентами компьютерной системы. Интерфейс необходим для того, чтобы пользователь мог проанализировать результаты работы программы и удостовериться в правильном её исполнении.

Модель пользовательского интерфейса – это представление того, как конечный пользователь взаимодействует с компьютерной программой или другим устройством, а также как реагирует система. Задача моделирования состоит в том, чтобы показать все "непосредственно испытываемые аспекты интерфейса или устройства". Модель интерфейса состоит из 5 компонентов, которые выполняют следующие функции:

1. Задать мощность множества
2. Обозначить в таблице с помощью столбцов структуры данных, с помощью которых будет представлено множество
3. Обозначить в таблице с помощью строк типы операций, которые будут выполнены над множеством
4. Отобразить время выполнения операции над множеством в определенной структуре данных
5. Закрыть окно с графическим интерфейсом

Модель интерфейса приложения курсового проекта представлена на рисунке 2.

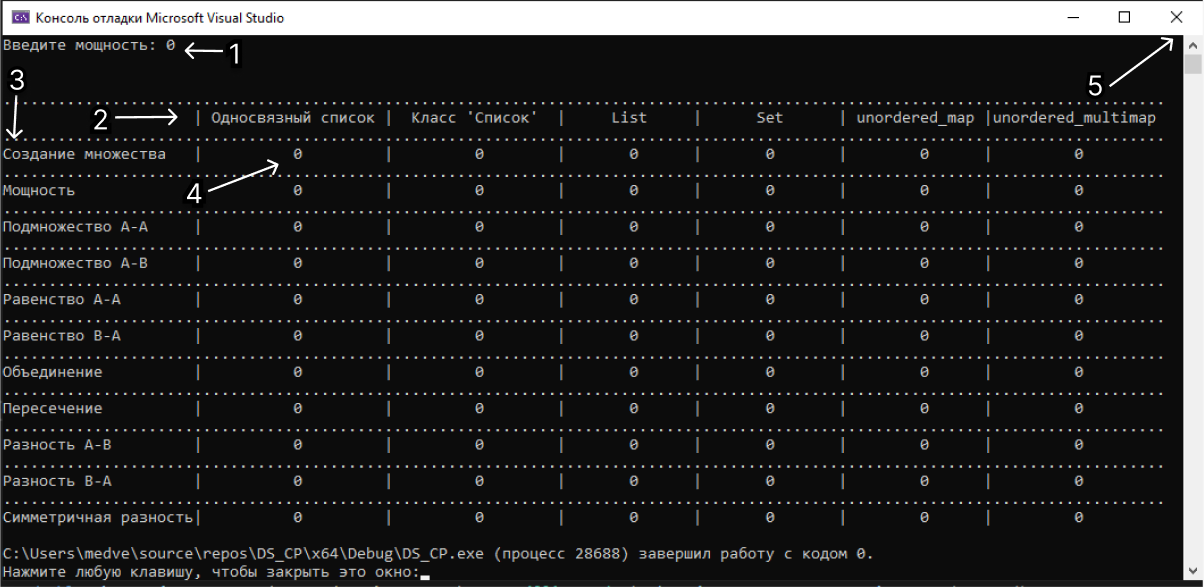


Рисунок 2 – Модель интерфейса

## Проектирование структур данных

В курсовой работе необходимо реализовать множество. Для этого используются следующие структуры данных: односвязный список, класс «Список» и контейнерные классы List, Set, Unordered\_map и Unordered\_multimap.

**Односвязный список** – это динамическая структура данных, состоящая из узлов. Каждый узел имеет какое-то значение, например, целое число и указатель на следующий узел.

Структура каждого узла представляет из себя:

struct Set {

int setElement; //значение

Set\* next; //его ячейка в памяти

};

Ниже приведено описание функции «Создать множество»:

Set\* createSet(int elemCount, int rangeDown, int rangeUp) {

if (elemCount < 0)

return NULL;

Set\* set = createEmptySet();

set = addElemInSet(elemCount, rangeDown, rangeUp);

int counter = 1;

while (counter < elemCount){

int localElement = set->setElement;

set = addElemInSet(elemCount, rangeDown, rangeUp);

if (localElement != set->setElement)

counter++;

}

return set;

}

Функция принимает параметры: количество элементов, нижняя и верхняя граница диапазона возможных значений этих элементов. Возвращает указатель на начало множества или, другими словами, на первый узел. Изначально создаётся пустое множество, а единственный узел указывает на NULL. Впоследствии добавляются новые узлы с помощью функции addElemInSet(), в которой автоматически соблюдается проверка уникальность элементов множества. Также для соблюдения уникальности необходим цикл while и переменная int localElement.

**Класс «Список»** – реализация односвязного списка, но с применением абстракций ООП. Он содержит ту же структуру узла списка, но помимо этого имеется поле, представленное ниже:

List\* head; //голова списка

Теперь методы будут автоматически знать какой указатель необходим, чтобы работать со списком в отличие от прошлого способа реализации списка, где необходимо было передавать указатель на начальный узел списка в параметры функции.

Описание функции «Определить множество функции»:

int SetList::power(){

int power = 0;

List\* localList = head;

while (localList != NULL){

power++;

localList = localList->next;

}

return power;

}

Создаётся локальный указатель на голову списка для дальнейшего прохода по узлам. Затем в цикле происходит проход с помощью указателей, а также одновременно подсчитывается количество элементов. В конце функция возвращает значение количества.

**Контейнерный класс «List»** – структура данных из библиотеки STL, позволяющая избавить нас от ручного написания односвязного списка. Она реализована разработчиками языка C++ и не требует глубоких знаний в реализации данной структуры данных. Вместо узлов, реализованных с помощью Struct, используются ячейки контейнера.

Описание «List» представляет из себя код:

list<int> list;

Методы, реализующие операции над множеством, реализованы с помощью встроенного функционала данного контейнера. Например, ниже представлено описание методы «Добавить новый элемент в множество»:

void SetByList::push\_front(int element){

if (!find(element)){

list.push\_front(element);

}

}

Изначально проверяется содержание элемента, который мы передаём в параметр, в нашем множестве. Если элемент отсутствует, с помощью встроенного метода контейнера «List» он добавляется в начало данной структуры данных.

**Контейнерный класс «Set»** – структура данных STL, по своему принципу схожая с «List», но созданная на основе бинарного дерева, а не односвязного списка. Каждый элемент является куском дерева, а при добавлении элементы автоматически становятся упорядоченными.

Ниже представлено описание контейнера «Set»:

set<int> bySet;

Также как и с контейнером «List» операции над множеством делаются с помощью встроенных методов в «Set». Операция «Проверить принадлежность элемента множеству» представлена ниже:

bool SetByList::find(int element) {

for (auto it = list.begin(); it != list.end(); it++){

if (\*it == element) {

return true;

}

}

return false;

}

С помощью итераторов в цикле, которые предоставляют доступ к элементам коллекции, можно пройтись по нашему множеству. Итератор it указывает на текущий элемент, чтобы получить доступ к этому элементу его нужны разыменовать. На каждой итерации происходит сравнение текущего элемента с элементом, который мы хотим проверить на наличие в множестве. Если такой элемент есть, метод возвращается true, иначе false.

**Unordered\_map** и **unordered\_multimap** –ассоциативные контейнеры, содержащие пару ключ: значение и реализующие хеш-таблицу, которая строится на основе ключей. Значения же автоматически с ними связываются, образуя пару.

Ниже представлен код описания данных структур:

unordered\_map<int, int> map;

unordered\_multimap<int, int> multimap;

По своей сути map и multimap работают одинаково, кроме того, что map хранит лишь одинаковые элементы, а multimap избавлен от этого. Но в связи с тем, что структура данных «Множество» по своей сути подразумевает хранение только уникальных элементов, фактически данные контейнеры не будут отличаться друг от друга. Поэтому имеет смысл описание метода одного из данных контейнеров. Данный метод использует встроенный функционал контейнерных классов STL. Описания метода «Разность множеств» представлен ниже:

SetByMap\* SetByMap::setDifference(SetByMap\* someSet){

SetByMap\* interSet = new SetByMap();

for (auto it = bySet.begin(); it != bySet.end(); ++it){

if (!someSet->find(\*it)) {

interSet->push\_front(\*it);

}

}

return interSet;

}

Данный метод своим параметром принимает множество, с которым будет найдена разность от множества, вызывающего данный метод. Так как **разностью** двух множеств называется множество, состоящее из тех и только тех элементов, которые входят в первое множество, но не входят во второе, необходимо создать третье множества. С помощью прохода по итераторам в цикле for находится элемент первого множества, не содержащийся во втором, а затем добавляется в наше третьей множество, которое будет возвращено в конце выполнения данного метода.

## Структура программного обеспечения

Для описания структуры программного обеспечения используются некоторые диаграммы. Одна из таких диаграмм – диаграмма классов. Она изображает множество классов, интерфейсов, коопераций и их отношений и используется для иллюстрации статического вида системы с точки зрения проектирования. Отношение «Ассоциация» необходимо чтобы показать, что определенный класс использует объекты других классов нашей программы.

На рисунке 3 представлена диаграмма классов курсового проекта.

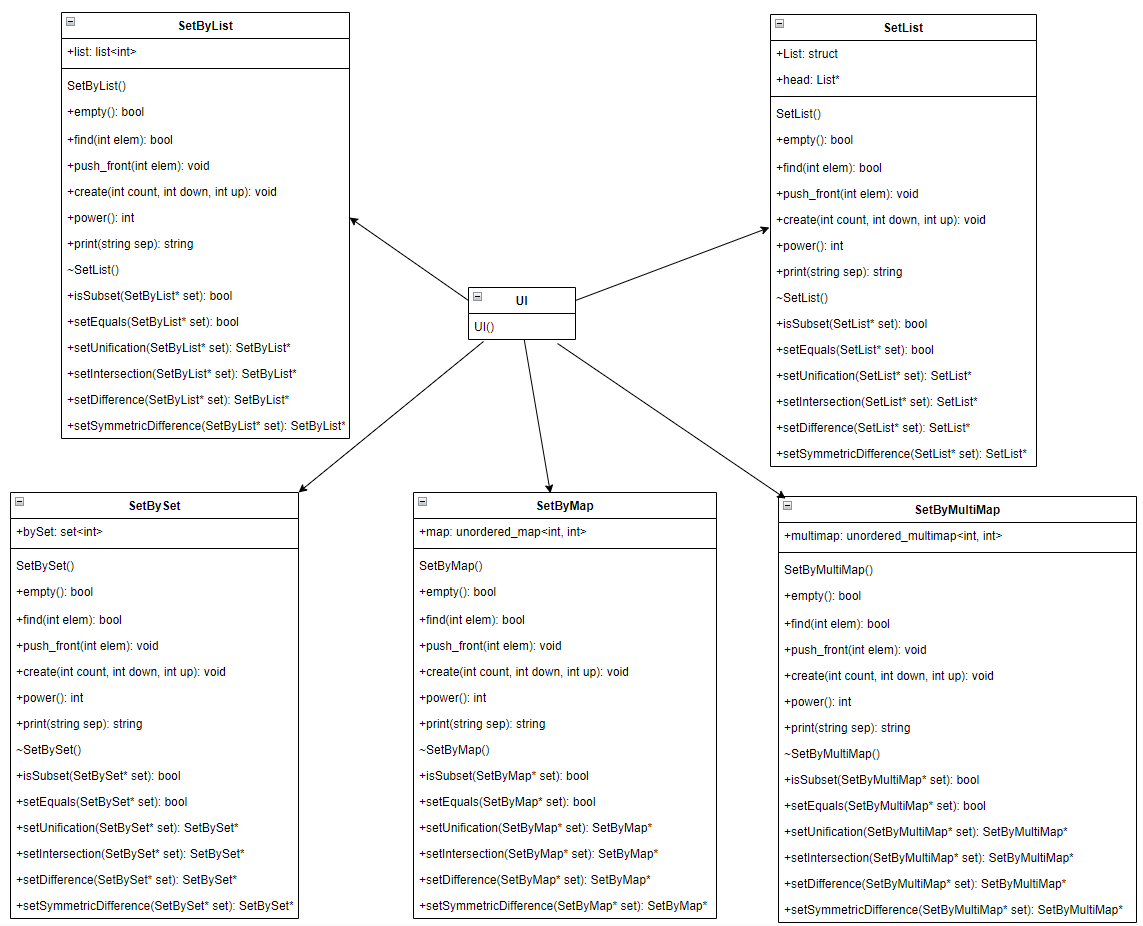


Рисунок 3 – Диаграмма классов

В классе UI инициализируется форма приложения курсового проекта для взаимодействия с ним, в нём описывается логика и функционал. Также здесь инициализируются все структуры данных, подсчитывается время выполнения операций, а итоги выводятся в консольное приложение. Этот класс ассоциативно связан с другими.

Остальные классы представляют из себя реализацию множества с помощью той или иной структуры данных, требуемой заданием курсовой работы.

Класс SetList используется для работы с классом «Cписок».

Класс SetByList используется для создания множества на основе контейнерного класса «List».

Класс SetBySet содержит в себе контейнер Set, который позволяет нам взаимодействовать с множеством.

Классы SetByMap и SetByMultiMap реализуют множество на основе хеш-таблиц unordered\_map и unordered\_multimap.

Данные классы, реализующие структуру данных «Множество» по своей структуре и операциям схожи друг с другом. Отличия заключаются лишь в том, что класс SetList использует поле Struct List, реализующее узлы одномерного массива, остальные же классы содержат в себе определенный контейнер, соответствующий названию класса. Методы данных классов по своей функциональности одинаковы, а по сигнатуре отличаются лишь типом возвращаемого значения. Поэтому целесообразно обратится лишь к описанию методов класса SetList. Оно находится в таблице 2 раздела 1.3.2 данной курсовой работы.

# 3. Реализация программы

## 3.1. Кодирование

В процессе выполнения курсовой работы было разработано приложение «Реализация структуры данных Множество», код которого приведен в приложении А.

Для алгоритма операции объединение, реализованной в качестве метода множества на основе контейнера «List», была разработана диаграмма деятельности, приведенная на рисунке 4.

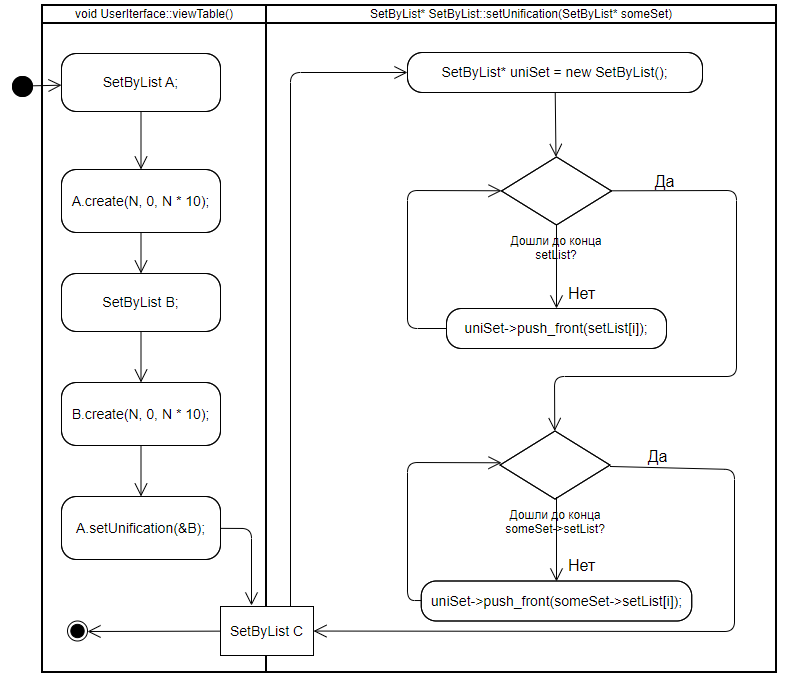


Рис 4 – Диаграмма деятельности

На диаграмме используются следующие переменные:

- int N – размер множеств;

- SetByList A, B, C – множества на основе контейнера «List»;

- list<int> setList – контейнерный класс «List», коллекция элементов исходного множества;

- SetByList\* uniSet – временное третье множество, являющееся результатом объединения множеств A и B;

## 3.2. Диаграмма компонентов

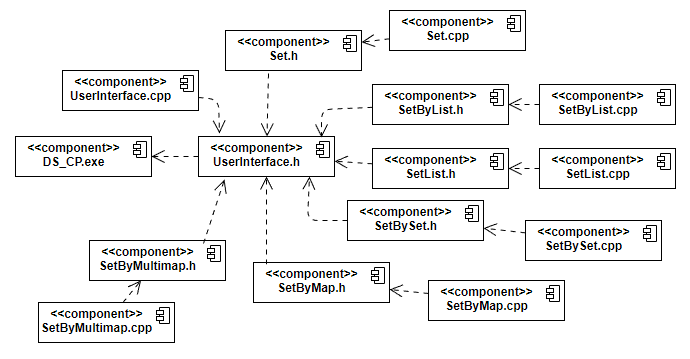
В процессе выполнения курсовой работы была составлена диаграмма компонентов на рисунке 5, которая отображает разбиение программной системы на структурные компоненты и связи между компонентами.

Рис 5 – Диаграмма компонентов

Описание компонентов приведено в таблице 8.

Таблица 8 – Компоненты для предметной области «Реализация структуры данных множество»

|  |  |
| --- | --- |
| **Компоненты** | **Назначение** |
| UserInterface.h | Заголовочный файл класса UserInterface |
| UserInterface.cpp | Исходный файл класса UserInterface |
| Set.h | Заголовочный файл класса Set |

Продолжение таблицы 8

|  |  |
| --- | --- |
| **Компоненты** | **Назначение** |
| Set.cpp | Исходный файл класса Set |
| SetList.h | Заголовочный файл класса SetList |
| SetList.cpp | Файл реализации класса SetList |
| SetByList.h | Заголовочный файл класса SetByList |
| SetByList.cpp | Исходный файл класса SetByList |
| SetBySet.h | Заголовочный файл класса SetBySet |
| SetBySet.cpp | Исходный файл класса SetBySet |
| SetByMap.h | Исходный файл класса SetByMap |
| SetByMap.cpp | Заголовочный файл класса SetByMap |
| SetByMultimap.h | Исходный файл класса SetByMultimap |
| SetByMultimap.cpp | Заголовочный файл класса SetByMultimap |
| DS\_CP.exe | Исполняемый файл приложения |