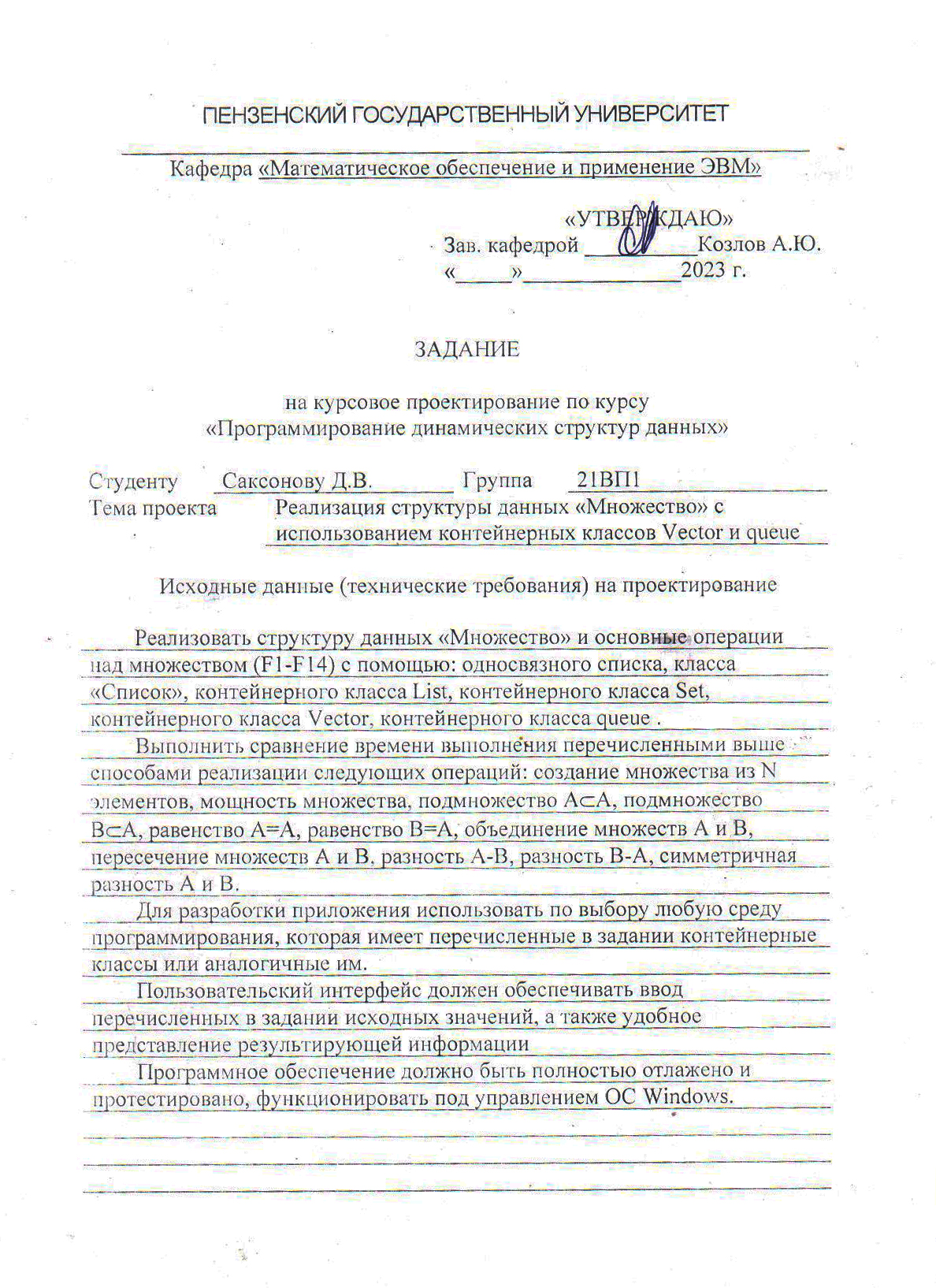
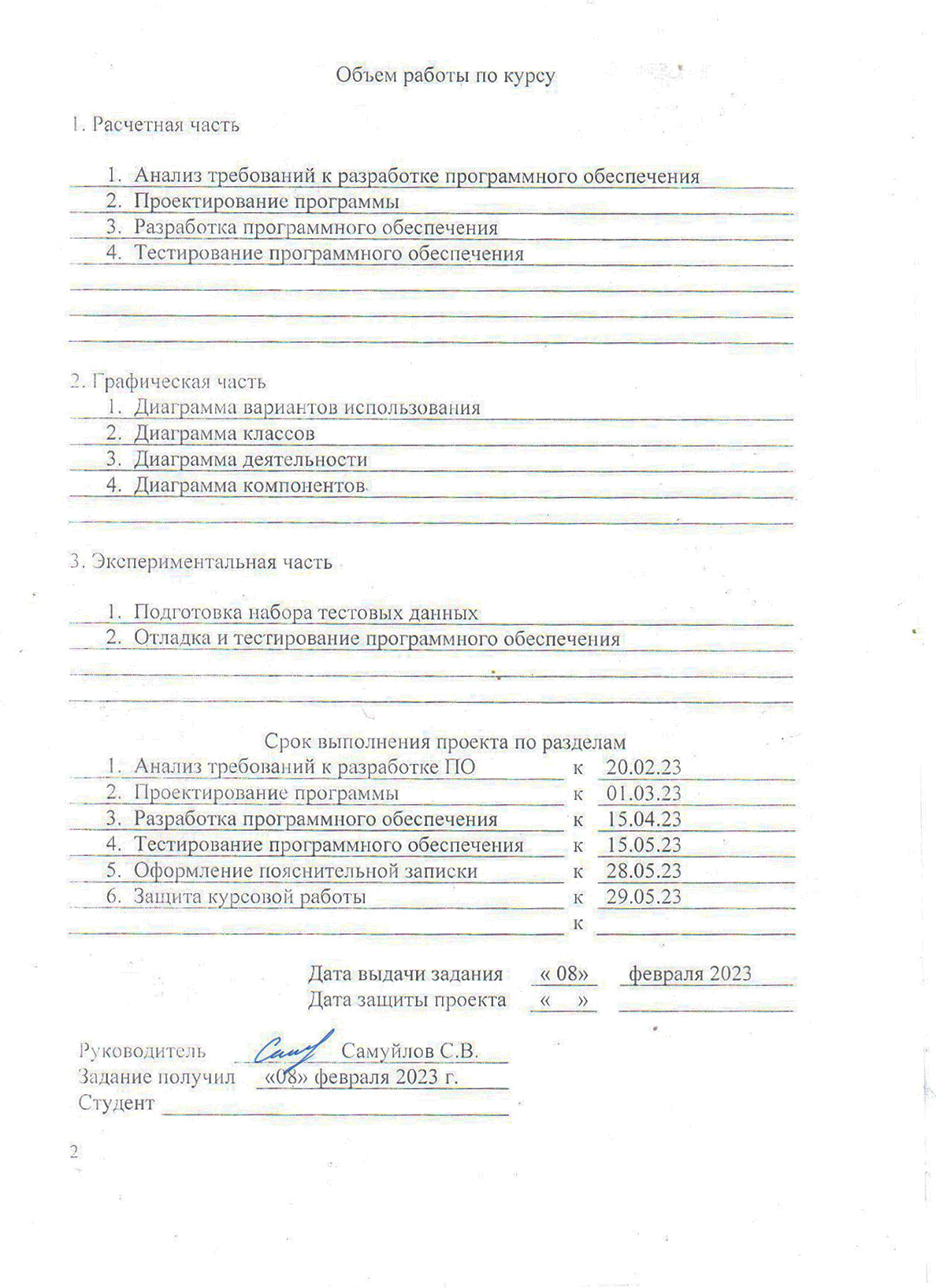
****

****

1. **Постановка задачи и анализ предметной области**
   1. **Основные понятия и определения**

Множество – совокупность каких-либо объектов, обладающих общим свойством. Эти объекты называют элементами множества.

Множество – это одно из ключевых понятий, используемых в математике, но оно также встречается и в информатике, этот объект используется в языках программирования. Это структура данных, которая представляет собой неупорядоченный набор уникальных элементов одного и того же типа. Элементами в множестве могут выступать строки, отдельные символы, числовые записи.

Можно выполнять различные действия не только с элементами множеств. Программирование позволяет управлять несколькими подобными множествами. Для этого применяются разнообразные операции.

Операции над множествами:

* Создание множества;
* Мощность множества: Мощность множества определяет количество элементов в множества;
* Подмножество: Множество A является подмножеством множества B, если все элементы множества A также являются элементами множества B. Обозначение: A ⊆ B;
* Равенство множеств: Множество A равно множеству B, если они содержат одни и те же элементы и равны по мощности. Обозначение: A = B;
* Объединение множеств: Объединение множеств A и B представляет собой множество, которое содержит все элементы множеств A и B. Обозначение: A ∪ B;
* Пересечение множеств: Пересечение множеств A и B представляет собой множество, которое содержит только те элементы, которые присутствуют в обоих множествах A и B. Обозначение: A ∩ B;
* Разность множеств: Разность множеств A и B представляет собой множество, которое содержит все элементы множества A, которые не присутствуют в множестве B. Обозначение: A \ B;
* Симметричная разность множеств: Симметричная разность множеств A и B представляет собой множество, которое содержит все элементы, которые присутствуют в A или B, но не в обоих. Обозначение: A Δ B;

Контейнерные классы – это универсальные шаблонные классы, предназначенные для хранения элементов заданного типа в смежных областях памяти. Стандарт C++ уже включает в себя большое количество контейнеров, как часть STL.

STL (Standart Template Library – Стандартная Библиотека Шаблонов) – набор согласованных [обобщённых](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [алгоритмов](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC), [контейнеров](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D0%B5%D0%B9%D0%BD%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), средств доступа к их содержимому и различных вспомогательных функций в [C++](https://ru.wikipedia.org/wiki/C%2B%2B).

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.

Контейнерные классы, которые используются в курсовой работе:

* Контейнерный класс List

Контейнер List представляет двухсвязный список, то есть такой список, где каждый элемент имеет указатели на предыдущий и последовательный элемент. Благодаря чему мы можем перемещаться по списку как вперед, так и назад. Для использования списка необходимо подключить заголовочный файл list.

В отличие от других контейнеров для типа list не определена операция обращения по индексу или функция, которая выполняет похожую задачу.

Тем не менее для контейнера list можно использовать функции front() и back(), которые возвращают соответственно первый и последний элементы. Следовательно, чтобы обратиться к элементам, которые находятся в середине, придётся выполнять перебор элементов.

* Контейнерный класс Set

Множество (set) представляет такой контейнер, который может хранить только уникальные значения. Как правило, множества применяются для создания коллекций, которые не должны иметь дубликатов. Множества представлены типом std::set<>, который определен в заголовочном файле <set>.

* Контейнерный класс Vector

Вектор представляет контейнер, который содержит коллекцию объектов одного типа. Для работы с векторами необходимо включить заголовок: <vector>.

* Контейнерный класс queue

Класс std::queue<T> представляет очередь - контейнер, который работает по принципу FIFO (first-in first-out или "первый вошел — первым вышел") — первым всегда извлекается первый добавленный элемент. То есть это контейнер, аналогичный стандартной очереди, которая часто встречается в нашей повседневной жизни.

Для работы с очередью надо подключать заголовочный файл <queue>.

* 1. **Постановка задачи**

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 Vector, контейнерного класса queue.

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

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 OC Windows.

* 1. **Анализ требований**
     1. **Требования к интерфейсу пользователя**

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

* вводить мощность каждого из множеств;
* 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 столбцов которой – структуры данных, а заголовки строк – наименования основных операций над ними.

* + 1. **Требования к структурам данных**

В курсовой работе используются следующие структуры данных: односвязный список, класс «Список», контейнерный класс List, контейнерный класс Set, контейнерный класс Vector, 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. |

Продолжение таблицы 1.

|  |  |
| --- | --- |
| Добавление узла в список | Функция добавления узла в список принимает два аргумента:  Указатель на узел, после которого происходит добавление;  Данные для добавляемого узла.  Если узел не передается, добавление нового узла происходит в начало списка. |
| Удаление узла из списка | В качестве аргументов функции удаления элемента ОЛС 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. |

Продолжение таблицы 1.

|  |  |
| --- | --- |
| Взаимообмен двух узлов списка | В качестве аргументов функция взаимообмена ОЛС принимает два указателя на обмениваемые узлы, а также указатель на корень списка. Функция возвращает адрес корневого элемента списка.  Взаимообмен узлов списка осуществляется путем переустановки указателей. Для этого необходимо определить предшествующий и последующий узлы для каждого заменяемого. Возможны две ситуации: заменяемые узлы – соседи и не соседи (между ними хотя бы один элемент). |

* 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 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 списка | 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 последовательный обход всех узлов с выводом их значений. |

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 функция принимает 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. |

* Контейнерный класс List

Объявление:

#include <list>

std::list <тип> имя;

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 |
| push\_front | Добавить элемент в начало |
| push\_back | Добавить элемент в конец |
| front | Обратиться к первому элементу |
| back | Обратиться к последнему элементу |
| insert | Добавить новый элемент в любое место |
| copy | Вывести все элементы списка |
| unique | Удалить все дубликаты |
| merge | Добавление другого списка |

* 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 | Указатель на начало контейнера |
| end | Указатель на конец контейнера |
| swap | Меняет содержимое контейнеров местами |
| insert | Вставка элемента |
| clear | Очистка контейнера |
| erase | Удаляет последовательность элементов |

Продолжение таблицы 4.

|  |  |
| --- | --- |
| count | Количество элементов в контейнере |
| find | Найти элемент в контейнере |

* Контейнерный класс Vector

Объявление:

#include <vector>

vector <тип> имя;

В таблице 5 представлены основные операции данной структуры данных.

Таблица 5 – Операции контейнерного класса Vector

|  |  |
| --- | --- |
| Название метода | Назначение |
| push\_back | Добавить элемент в начало |
| pop\_back | Удалить последний элемент вектора |
| clear | Удалить все элементы вектора |
| insert | Добавить элемент на определенную позицию |
| size | Узнать размер вектора |
| resize | Изменить размер вектора |
| assign | Заменить все элементы вектора |
| swap | Обменивает значения двух контейнеров |

* Контейнерный класс queue

Объявление:

#include <queue>

std::queue <тип> имя;

В таблице 6 представлены основные операции данной структуры данных.

Таблица 6 – Операции контейнерного класса queue

|  |  |
| --- | --- |
| Название метода | Назначение |
| size | Получить количество элементов в очереди |
| empty | Проверить очередь на наличие элементов |
| push | Добавить элемент в очередь |
| front | Получить первый элемент очереди |
| back | Получить последний элемент в очереди |
| pop | Удалить элемент из начала |
| swap | Обменять элементы двух очередей |

* + 1. **Требования к программным средствам**

Разрабатываемая программа должна выполнять следующие функции:

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

Необходимые функции разрабатываемой программы отображены на диаграмме вариантов использования, которая представлена на рисунке 1.

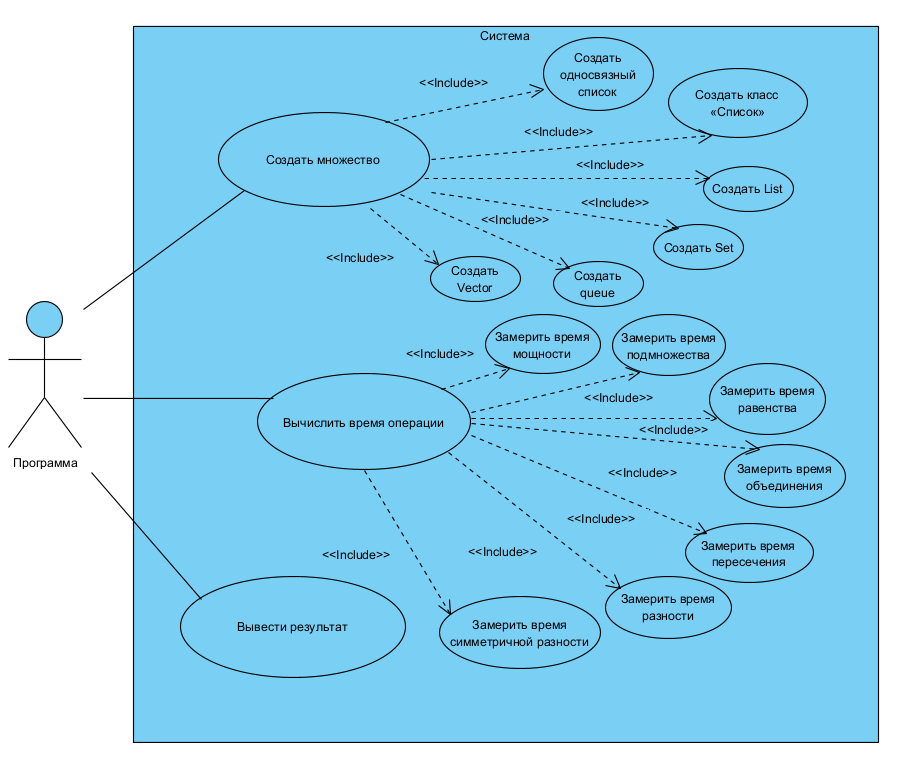


Рисунок 1 – Диаграмма вариантов использования

Описание спецификации прецедентов приведены в таблицах 7 – 9.

Таблица 7 – Сценарий варианта использования «Нахождение мощности множества»

|  |
| --- |
| **Наименование: Нахождение мощности множества** |
| **ID: 1** |
| **Краткое описание:** система находит мощность множества |
| **Действующие лица:** пользователь |
| **Основной поток:**   1. Система вычисляет количество элементов в множестве |
| **Постусловие:** мощность множества найдена |

Таблица 8 – Сценарий варианта использования «Объединение множества A и B»

|  |
| --- |
| **Наименование: Объединение множества A и B** |
| **ID: 2** |
| **Краткое описание:** система создает множество, состоящее из объединения множеств A и B |
| **Действующие лица:** пользователь |
| **Предусловие:** создано два множества A и B |
| **Основной поток:**  1.Система создает одно множество из элементов A и B, не включая повторяющиеся элементы |
| **Постусловие:** создано множество, содержащее объединение двух множеств |

Таблица 9 – Сценарий варианта использования «Симметричная разность A и B»

|  |
| --- |
| **Наименование: Симметричная разность A и B** |
| **Краткое описание:** система создаёт множество, состоящее из симметричной разности множеств A и B |
| **Действующие лица:** пользователь |
| **Предусловие:** создано два множества A и B |
| **Основной поток:**  1. Система создает множество, состоящее из объединения двух множеств A и B  2. Система создает множество, состоящее из пересечения двух множеств A и B  3. Система создает множество, состоящее из разности объединения и пересечения двух множеств A и B |
| **Постусловие:** создано множество, состоящее из симметричной разности двух множеств |

