**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 forward\_list и unordered\_map»

ПГУ 09.03.04 – 04КР212.03 ПЗ

Направление подготовки – 09.03.04 Программная инженерия

Профиль подготовки – Программная инженерия

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

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

2023

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как стол

Автоматически созданное описание

# **1. Постановка задачи и анализ предметной области**

## **1.1. Основные понятия и определения**

Множество – совокупность объектов, объединённых по какому-либо признаку.Под множеством понимается всё то многое, которое можно мыслить, как единое, то есть такую совокупность определённых элементов, которая посредством одного закона может быть соединена в одно целое.

Элементы множества – это объекты, которые образуют данное множество, и могут обладать некоторыми свойствами и находиться в некоторых отношениях между собой или с элементами других множеств. Элементы множества различны и отличимы друг от друга.

Конечное множество ­– множество, которое содержит конечное число элементов.

Бесконечное множество – множество, содержащее бесконечное количество элементов.

Пустое множество – множество, не содержащее ни одного элемента.

Подмножество – множество A является подмножеством множества B, если каждый элемент A также является элементом B.

Свойства множества:

* Каждый элемент совокупность уникален, то есть отличим от других элементов
* Для любого объекта существует возможность установить принадлежит ли он множеству или нет

При работе с множествами часто используются следующие обозначения:

* ⇒ импликация (следование) А ⇒ В, если А верно, то В тоже верно
* <=> (равносильно) А верно тогда и только тогда, когда В верно
* | – таких что, { x | P(x) } — множество всех таких что, верно Р(х)
* & – обозначение связки “и”
* | — обозначение “или”
* Квантор общности
* Квантор существования

Основные операции над множествами:

* Создание множества – операция, которая создает множество из N элементов
* Мощность – операция |A|, позволяющая узнать мощность множества.
* Подмножество– операция А ⊂ В, проверяющая является ли множество А подмножеством В.
* Равенство – операция А = В, проверяющая идентичность двух множеств.
* Дополнение – операция B = , результатом которой является множество, в которое входят все элементы первого множества, не входящие во второе множество
* Объединение – операция А∪В, результатом которой является множество, содержащее в себе все элементы исходных множеств.
* Пересечение – операция А∩B, результатом которой является множество, содержащий элементы присутствующие сразу в обоих множествах.
* Разность – операция А\B, результатом которой является множество, элементы которого принадлежат А, но не принадлежат В.
* Симметричная разница – операция А Δ В, результатом которого является множество, каждый элемент которого принадлежит или А, или В, но не обоим сразу.

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 (Standard Template Library -- Стандартная Библиотека Шаблонов). Данные хранятся в контейнерах, а операции с ними определяются методами контейнеров и адаптируемыми алгоритмами. Итераторы «склеивают» эти два компонента. Благодаря им любой алгоритм может работать с любым контейнером.

Последовательные контейнеры обеспечивают хранение конечного количества однотипных величин в виде непрерывной последовательности. К ним относятся векторы (vector), двусторонние очереди (deque), списки (list) и односвязные списки (forward\_list), а также так называемые адаптеры, то есть варианты контейнеров, включая стеки (stack), очереди (queue) и очереди с приоритетами (priority\_queue). Еще одним видом контейнера с ограниченным набором операций является массив(array).

Ассоциативные контейнеры обеспечивают быстрый доступ к данным по ключу. Эти контейнеры построены на основе сбалансированных деревьев. Существует пять типов ассоциативных контейнеров: словари (map), словари с дубликатами (multimap), 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. Например, при необходимости часто вставлять и удалять элементы в середине последовательности следует использовать списки, а если включение элементов выполняется главным образом в конец или начало — двустороннюю очередь.

Контейнерный класс List представляет собой двусвязный список, где каждый элемент имеет указатели на предыдущий и следующий элементы. Он предоставляет эффективный способ добавления, удаления и перемещения элементов в списке, в то время как доступ к произвольному элементу происходит за линейное время. Класс List предоставляет множество методов для работы со списками, таких как push\_back(), push\_front(), pop\_back(), pop\_front(), insert(), erase() и т.д.

Контейнерный класс Set представляет собой отсортированное множество уникальных элементов. Элементы внутри множества автоматически сортируются по возрастанию и не могут иметь дубликатов. Это особенно удобно для хранения данных, когда важен только факт присутствия элемента, но не его порядок. Класс Set предоставляет методы для работы с множеством, такие как insert(), erase(), find() и т.д.

Класс forward\_list реализует односвязный список, элементы которого хранят указатель только на следующий элемент (в списках класса list элементы хранят указатели и на предыдущий, и на последующий элементы). Вставка и удаление элементов происходит быстро, достаточно изменить ссылку, но получить доступ к элементу списка по индексу нельзя, т. к. элементы могут быть расположены в разных местах памяти. Перебирать элементы можно только в прямом направлении: от первого элемента до последнего. В некоторых случаях списки класса forward\_list могут быть эффективнее списков класса list, но они менее универсальны.

Класс unordered\_map реализует ассоциативный массив, в котором каждому ключу соответствует одно значение. Чтобы получить доступ к элементу, необходимо указать ключ, который использовался при сохранении элемента. В отличии от класса map для быстрого поиска элементов используется хеш-таблица, значения которой состоят из целых чисел (тип size\_t).

## **1.2. 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 forward\_list, контейнерного класса unordered\_map.

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

Интерфейс пользователя должен удобно представлять информацию о мощности каждого множества, а также выводить в виде таблицы время выполнения различными методами следующие операции над множествами:

* создание множества из N элементов,
* мощность множества, подмножество А ⊂ A,
* подмножество В ⊂ А,
* равенство А = А,
* равенство В = А,
* объединение множеств А и В,
* пересечение множеств А и В,
* разность А - В, разность B - A,
* симметричная разность А и В.

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.

### **1.3.2. Требования к структурам данных**

В курсовой работе используются следующие структуры данных:

* односвязный список,
* класс «Список»,
* контейнерный класс List,
* контейнерный класс Set,
* контейнерный класс forward\_list,
* контейнерный класс unordered\_map.

Объявление односвязного списка:

Set\* set = createNewSet(6 + rand() % 3, 10, 100)

Основные операции для работы с односвязным списком представлены в таблице 1.

Таблица 1 – основные методы над односвязным списком

|  |  |
| --- | --- |
| **Операция** | **Описание операции** |
| Set\* createEmptySet(); | Создание пустого множества |
| bool isEmptySet(Set\* first); | Проверка на пустоту множества |
| bool isSetHasElement(Set\* first, int element); | Проверка элемента на принадлежность множеству |
| Set\* addNewElement(Set\* first, int element); | Добавление нового элемента в начало множества |
| Set\* createNewSet(int quantity, int min, int max); | Создание множества по заданным параметрам |
| int setPowers(Set\* first); | Мощность множества |
| string setView(Set\* first, char separator); | Вывод элементов множества |
| Set\* deleteSet(Set\* first); | Удаление множества (очистка занимаемой множеством памяти) |
| bool isSubSet(Set\* first, Set\* second); | Подмножество А-B |
| bool isSetsEquals(Set\* first, Set\* second); | Равенство двух множеств А-В |
| Set\* unionOfSets(Set\* first, Set\* second); | Объединение двух множеств |
| Set\* intersectionsOfSets(Set\* first, Set\* second); | Пересечение двух множеств |
| Set\* differenceOfSets(Set\* first, Set\* second); | Разность множеств |
| Set\* symmetricDifferenceOfSets(Set\* first, Set\* second); | Симметричная разность |

Объявление класса «Список»

ListSet\* listSet = new ListSet(6 + rand() % 3, 10, 100)

Основные методы класса «Список» представлены в таблице 2.

Таблица 2 – методы класса «Список»

|  |  |
| --- | --- |
| **Метод** | **Описание метода** |
| bool EmptyListSet(); | Проверка на пустое множество |
| bool ItemInListSet(int value); | Проверяет принадлежности элемента множеству |
| void AddElement(int value); | Добавляет нового элемента в множество |
| int PowerListSet() | Определяет мощность множества |
| string ListSetView(char sym) | Выводит элементы множества с символьным разделителем |
| bool isSubSet(SetList\* sub\_set) | Проверка на подмножество |
| bool isEqualsSets(SetList\* second\_set) | Равенство двух множеств |
| SetList\* MergeSets(SetList\* second\_set) | Объединение двух множеств. |
| SetList\* IntersectionSets(SetList\* second\_set) | Пересечение двух множеств. |
| SetList\* DifferenceSets(SetList\* second\_set) | Разность множеств. |
| SetList\* SymmetricDifferenceSets(SetList\* second\_set); | Симметричная разность. |

Пример объявления контейнерного класса «List»:

list<int> myList;

Основные операции контейнерного класса «List» представлены в таблице 3.

Таблица 3 – основные операции 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 |

Продолжение таблицы 3.

|  |  |
| --- | --- |
| 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;

Основные операции над контейнерным классом «Set» представлены в таблице 4.

Таблица 4 – основные операции 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 |

Объявление контейнерного класса «forward\_list»:

forward\_list<int> numbers;

Основные операции над контейнерным классом «forward\_list» представлены в таблице 5.

Таблица 5 – основные операции контейнерного класс «forward\_list»

|  |  |
| --- | --- |
| **Метод** | **Описание метода** |
| push\_front(value) | Добавляет элемент в начало прямого списка |
| pop\_front() | Удаляет элемент в начале прямого списка. |
| empty() | Проверяет, пуст ли прямой список. |

Продолжение таблицы 5.

|  |  |
| --- | --- |
| max\_size () | Возвращает максимальную длину прямого списка. |
| remove(value) | Удаляет из прямого списка элементы, совпадающие с заданным значением. |

Объявление контейнерного класса «unordered\_map»:

unordered\_map<char, int> Mymap;

Основные операции над контейнерным классом «unordered\_map» представлены в таблице 6.

Таблица 6 – основные операции контейнерного класс «unordered\_map»

|  |  |
| --- | --- |
| **Метод** | **Описание метода** |
| insert(key, value) | Добавляет элементы. |
| erase(key) | Удаляет элементы в указанных позициях. |
| size() | Подсчитывает количество элементов. |
| empty() | Проверяет отсутствие элементов. |
| find(key) | Определяет элемент, соответствующий указанному ключу. |
| clear() | Удаляет все элементы. |

### **1.3.3. Требования к программным средствам**

Программа должна позволять пользователю вводить мощность множеств, а именно единую размерность односвязного списка, класса «Список», контейнерного класса «List», контейнерного класса «Set», контейнерного класса «forward\_list», контейнерного класса «unordered\_map». Вычислить время операций всех выше раннее перечисленных структур данных. Для сравнения времени выполнения операций выводить результаты в табличной форме, где заголовки столбцов является наименованием структур данных, а заголовки строк – наименование операций.

Диаграмма вариантов использования (ДВИ) – диаграмма, отражающая отношения между актерами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном Изображение выглядит как диаграмма

Автоматически созданное описаниеуровне.

На основе вышеперечисленного функционала была сформирована диаграмма вариантов использования (Рисунок 1).

Рисунок 1 – Диаграмма вариантов использования

Описания сценариев вариантов использования, а именно «Ввести мощность множеств», «Вычислить время операций», «Вывести результаты в табличном виде» приведены в таблицах 7–9.

Таблица 7 – Сценарий варианта использования «Ввести мощность множеств»

|  |
| --- |
| **Наименование: ввести размер множеств** |
| **Краткое описание:** система запрашивает у пользователя одно значение мощности для всех множеств |
| **Действующие лица:** пользователь |
| **Предусловие:** - |
| **Основной поток:**   1. Пользователь вводит мощность для всех множеств |
| **Постусловие:** - |

Таблица 8 – Сценарий варианта использования «Вычислить время операций»

|  |
| --- |
| **Наименование: вычислить время операций контейнерного класса “List”** |
| **Краткое описание:** система вычисляет время операций над множествами |
| **Действующие лица:** пользователь |
| **Предусловие: -** |
| **Основной поток:**   1. В функцию передается мощность множества N, полученную от пользователя 2. Функция формирует множество размерностью N 3. Функция вычисляет время, затраченное на выполнения операций |
| **Постусловие:** вычислено время выполнения операций |

Таблица 9 – Сценарий варианта использования «Вывести результаты в табличном виде»

|  |
| --- |
| **Наименование: вывести результаты в табличном виде** |
| **Краткое описание:** система выводит полученные результаты времени в табличном виде |
| **Действующие лица:** пользователь |
| **Предусловие:** результаты времени выполнения операций вычислены |
| **Основной поток:**   1. Система выводит результаты в табличном виде |
| **Постусловие:** результаты выведены в табличном виде |