МИНИСТЕРСТВО НАУКИ И ВЫСШЕГООБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

ЛАБОРАТОРНАЯ РАБОТА

на тему:

**«ОРГАНИЗАЦИЯ ДОСТУПА ПО ИМЕНИ (ТАБЛИЦЫ)»**

**Выполнил:** студент группы 3822Б1ФИ1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Созонов И.С. /

Подпись

**Проверил:** к.т.н., доцент каф. ВВиСП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Кустикова В.Д. /

Подпись

Нижний Новгород  
2024

**Содержание**

[Введение 3](#_Toc168422965)

[1 Постановка задачи 4](#_Toc168422966)

[2 Руководство пользователя 5](#_Toc168422967)

[2.1 Приложение для демонстрации работы таблиц 5](#_Toc168422968)

[3 Руководство программиста 19](#_Toc168422969)

[3.1 Использованные алгоритмы 19](#_Toc168422970)

[3.1.1 Неупорядоченные таблицы 19](#_Toc168422971)

[3.1.2 Упорядоченные таблицы 20](#_Toc168422972)

[3.1.3 Хэш-таблицы 23](#_Toc168422973)

[3.2 Описание классов 26](#_Toc168422974)

[3.2.1 Схема наследования классов 26](#_Toc168422975)

[3.2.2 Структура TabRecord 26](#_Toc168422976)

[3.2.3 Класс Table 27](#_Toc168422977)

[3.2.4 Класс ScanTable 29](#_Toc168422978)

[3.2.5 Класс SortedTable 30](#_Toc168422979)

[3.2.6 Класс HashTable 32](#_Toc168422980)

[3.2.7 Класс ArrayHashTable 32](#_Toc168422981)

[Заключение 36](#_Toc168422982)

[Литература 37](#_Toc168422983)

[Приложения 38](#_Toc168422984)

[Приложение А. Реализация структуры TabRecord 38](#_Toc168422985)

[Приложение Б. Реализация класса Table 38](#_Toc168422986)

[Приложение В. Реализация класса ScanTable 38](#_Toc168422987)

[Приложение Г. Реализация класса SortedTable 39](#_Toc168422988)

[Приложение Д. Реализация класса HashTable 41](#_Toc168422989)

[Приложение Е. Реализация класса ArrayHashTable 41](#_Toc168422990)

# Введение

Представление данных во многих задачах из разных областей человеческой деятельности может быть организовано при помощи таблиц. Таблицы представляют собой последовательности строк (записей), структура строк может быть различной, но обязательным является поле, задающее имя (ключ) записи. Таблицы применяются в бухгалтерском учете (ведомости заработной платы), в торговле (прайс-листы), в образовательных учреждениях (экзаменационные ведомости) и являются одними из наиболее распространенных структур данных, используемых при создании системного и прикладного математического обеспечения. Таблицы широко применяются в трансляторах (таблицы идентификаторов) и операционных системах, могут рассматриваться как программная реализация ассоциативной памяти и т.п. Существование отношения «иметь имя» является обязательным в большинстве разрабатываемых программистами структур данных; доступ по имени в этих структурах служит для получения соответствия между адресным принципом указания элементов памяти ЭВМ и общепринятым (более удобным для человека) способом указания объектов по их именам.

Целью лабораторной работы помимо изучения способов организации таблиц является начальное знакомство с принципами проектирования структур хранения, используемых в методах решения прикладных задач. На примере таблиц изучаются возможность выбора разных вариантов структур хранения, анализ их эффективности и определения областей приложений, в которых выбираемые структуры хранения являются наиболее эффективными.

В качестве практической задачи, на примере которой будут продемонстрированы возможные способы организации таблиц, рассматривается хранение полиномов в качестве записей таблицы.

# Постановка задачи

Цель – реализовать классы для представления таблиц различных видов, в которых можно хранить полиномы.

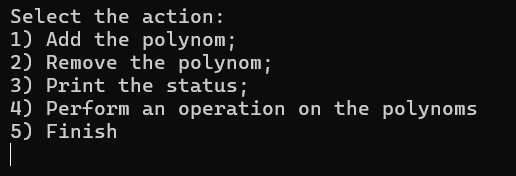
Задачи:

1. Разработать шаблонную структуру TabRecord для работы с записями таблицы.
2. Реализовать шаблонный класс Table для представления таблиц общего вида. Он должен поддерживать следующие основные операции: поиск записи по ключу, вставка записи в таблицу и удаление записи из таблицы. Также следует добавить специфические операции (проверку на полноту таблицы, проверку на пустоту таблицы, проверку, достигли ли конца таблицы) и навигационные методы (получение текущей записи, переход к первой записи, переход к следующей записи таблицы).
3. На основании структуры записи реализовать шаблонный класс ScanTable (наследник класса Table) для работы с просматриваемыми таблицами. Переопределить основные операции над таблицами.
4. Создать шаблонный класс SortedTable (наследник класса ScanTable) для работы с упорядоченными таблицами. Для упорядочивания записей в таблице реализовать метод быстрой сортировки и переопределить основные операции над таблицами.
5. Реализовать шаблонный класс HashTable (наследник класса Table) для работы с хэш-таблицами. Для отображения значения ключа в индекс строки разработать функцию хеширования.
6. На основании структуры записи создать шаблонный класс ArrayHashTable (наследник класса HashTable) для работы с хэш-таблицами, реализованными с помощью метода открытого перемешивания. Переопределить основные операции над таблицами.

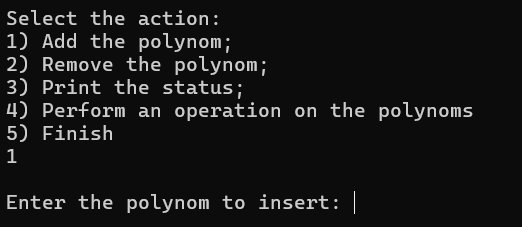
# Руководство пользователя

## Приложение для демонстрации работы таблиц

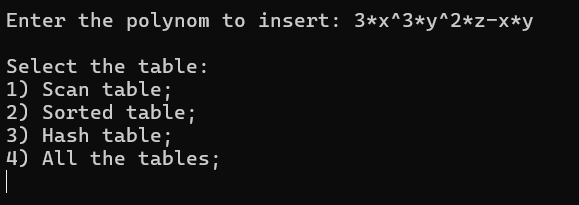
1. Запустить sample\_tables.exe. В результате появится окно для выбора действия: 1) добавить полином в таблицу; 2) удалить полином из таблицы; 3) вывести состояние таблиц; 4) выполнить операцию над полиномами; 5) завершить (рис. 1).



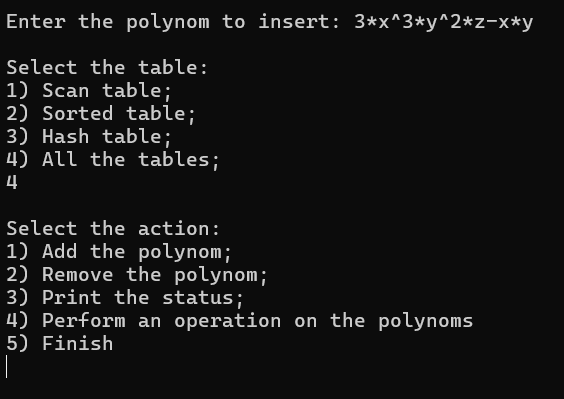
1. Основное окно приложения
2. Выбрать действие. Например, добавить полином в таблицу. В результате появится окно для ввода полинома, который необходимо поместить в таблицу (рис. 2).



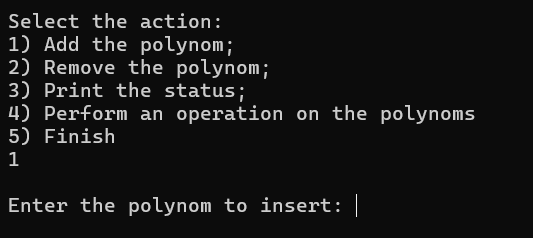
1. Выбор действия
2. Ввести полином. В результате появится окно для выбора таблицы, в которую необходимо вставить данный полином: 1) просматриваемая таблица; 2) упорядоченная таблица; 3) хэш-таблица; 4) все таблицы (рис. 3).



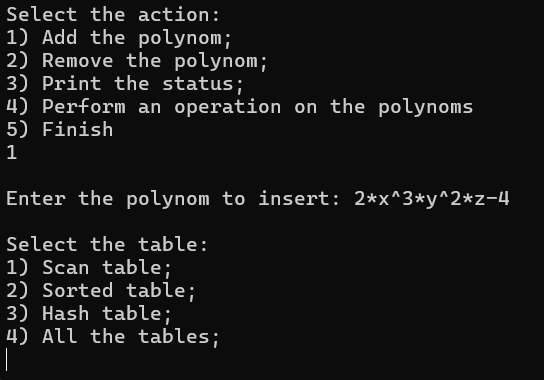
1. Ввод полинома, который небходимо поместить в таблицу
2. Выбрать таблицу. Например, все таблицы. В результате полином будет помещен во все таблицы и появится окно для выбора следующего действия (рис. 4).



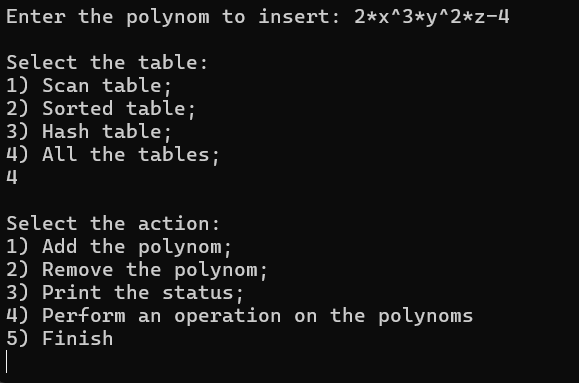
1. Выбор таблицы, в которую необходимо вставить полином
2. Выбрать действие. Например, вновь добавить полином в таблицу. В результате появится окно для ввода полинома, который необходимо поместить в таблицу (рис. 5).



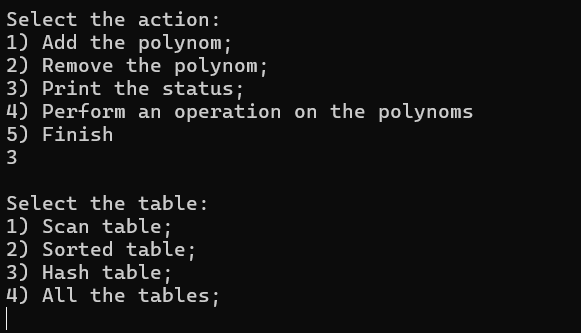
1. Выбор действия
2. Ввести полином. В результате появится окно для выбора таблицы, в которую необходимо вставить данный полином (рис. 6).



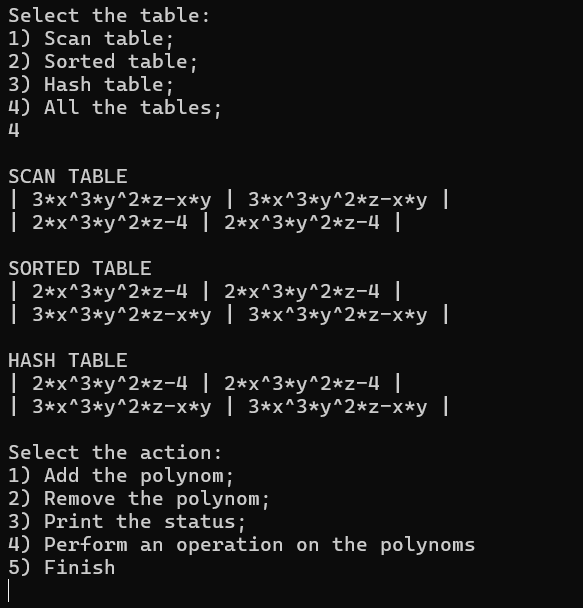
1. Ввод полинома, который небходимо поместить в таблицу
2. Выбрать таблицу. Например, все таблицы. В результате полином будет помещен во все таблицы и появится окно для выбора следующего действия (рис. 7).



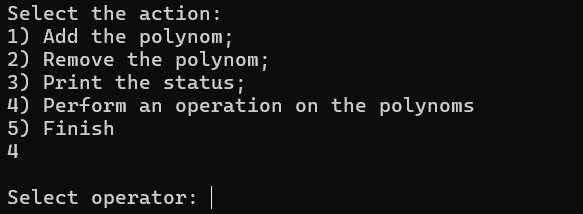
1. Выбор таблицы, в которую необходимо вставить полином
2. Выбрать действие. Например, вывести состояние таблиц. В результате появится окно для выбора таблицы, которую необходимо вывести (рис. 8).



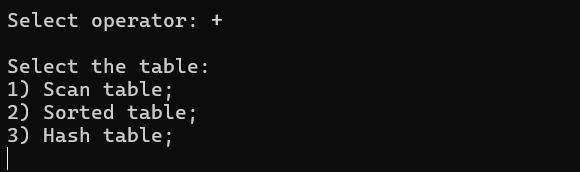
1. Выбор действия
2. Выбрать таблицу. Например, все таблицы. В результате будут выведены состояния всех таблиц и появится окно для выбора следующего действия (рис. 9).



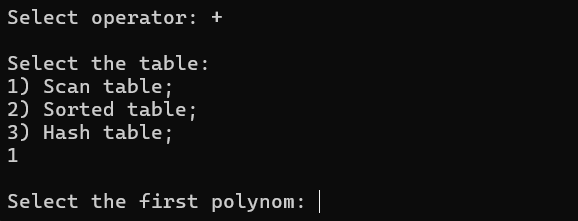
1. Выбор таблицы, которую необходимо вывести
2. Выбрать действие. Например, выполнить операцию над полиномами. В результате появится окно для ввода оператора данной операции (сложения, вычитания или произведения) (рис. 10).



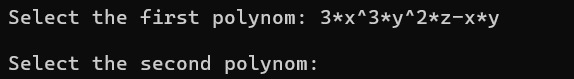
1. Выбор действия
2. Ввести оператор. Например, плюс. В результате появится окно для выбора таблицы, в которой производится сложение полиномов (рис. 11).



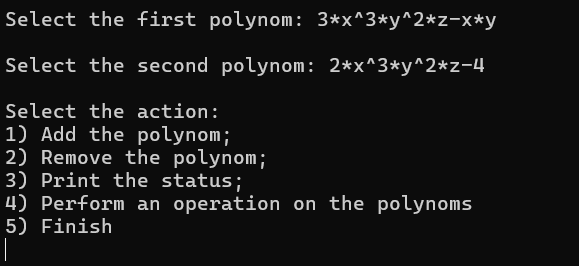
1. Ввод оператора
2. Выбрать таблицу. Например, просматриваемую. В результате появится окно для ввода первого полинома, к которому необходимо прибавить второй полином (рис. 12).



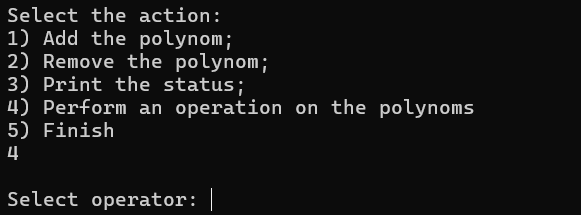
1. Выбор таблицы
2. Ввести первый полином. В результате появится окно для ввода второго полинома, который необходимо прибавить к первому полиному (рис. 13).



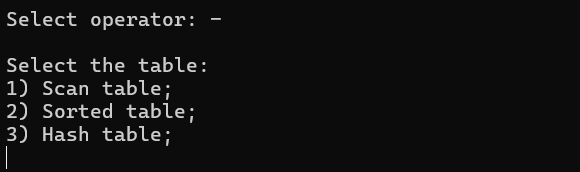
1. Ввод первого полинома
2. Ввести второй полином. В результате будет выполнено сложение полиномов, и их сумма будет помещена в просматриваемую таблицу. Затем появится окно для выбора следующего действия (рис. 14).



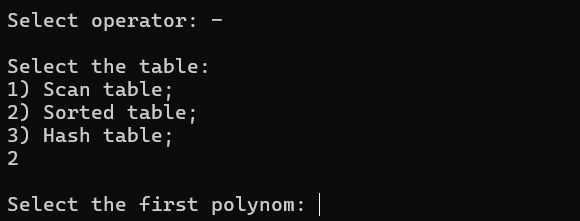
1. Ввод второго полинома
2. Выбрать действие. Например, выполнить операцию над полиномами. В результате появится окно для ввода оператора данной операции (рис. 15).



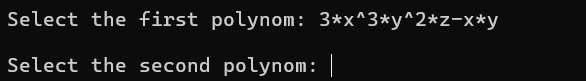
1. Выбор действия
2. Ввести оператор. Например, минус. В результате появится окно для выбора таблицы, в которой производится вычитание полиномов (рис. 16).



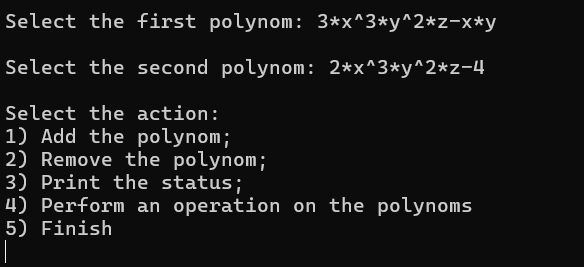
1. Ввод оператора
2. Выбрать таблицу. Например, упорядоченную. В результате появится окно для ввода первого полинома, из которого необходимо вычесть второй полином (рис. 17).



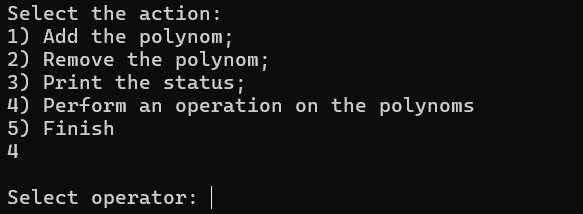
1. Выбор таблицы
2. Ввести первый полином. В результате появится окно для ввода второго полинома, который необходимо вычесть из первого полинома (рис. 18).



1. Ввод первого полинома
2. Ввести второй полином. В результате будет выполнено вычитание полиномов, и их разность будет помещена в упорядоченную таблицу. Затем появится окно для выбора следующего действия (рис. 19).



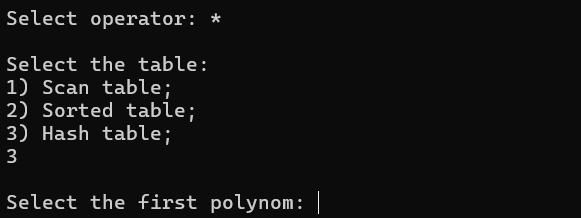
1. Ввод второго полинома
2. Выбрать действие. Например, выполнить операцию над полиномами. В результате появится окно для ввода оператора данной операции (рис. 20).



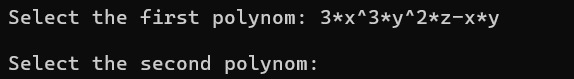
1. Выбор действия
2. Ввести оператор. Например, умножить. В результате появится окно для выбора таблицы, в которой производится умножение полиномов (рис. 21).



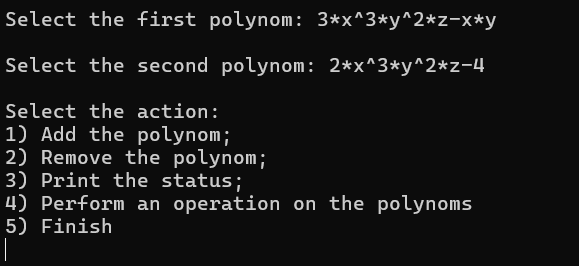
1. Ввод оператора
2. Выбрать таблицу. Например, хэш-таблицу. В результате появится окно для ввода первого полинома, на который необходимо умножить второй полином (рис. 22).



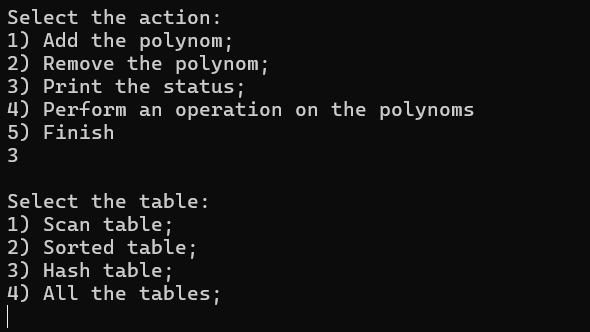
1. Выбор таблицы
2. Ввести первый полином. В результате появится окно для ввода второго полинома, который необходимо умножить на первый полином (рис. 23).



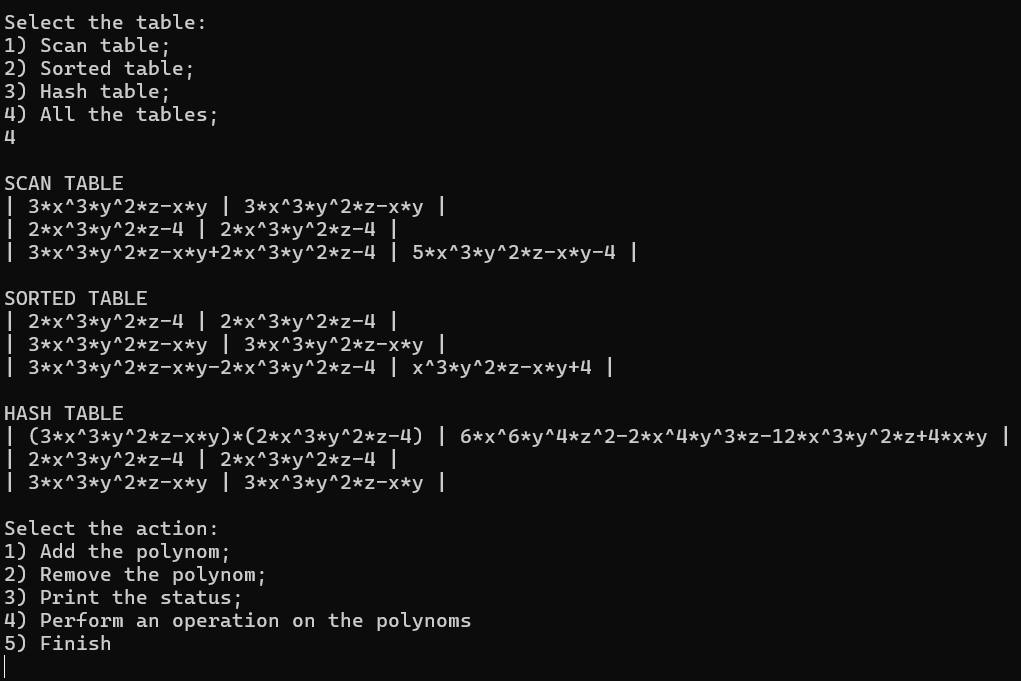
1. Ввод первого полинома
2. Ввести второй полином. В результате будет выполнено умножение полиномов, и их произведение будет помещено в хэш-таблицу. Затем появится окно для выбора следующего действия (рис. 24).



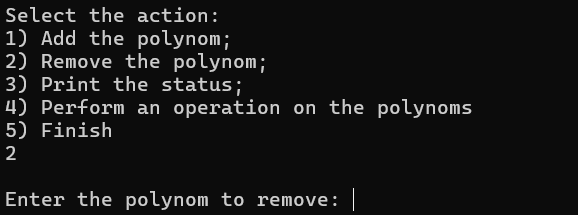
1. Ввод второго полинома
2. Выбрать действие. Например, вывести состояние таблиц. В результате появится окно для выбора таблицы, которую необходимо вывести (рис. 25).



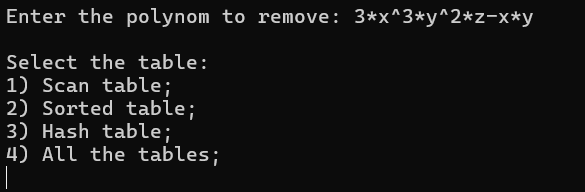
1. Выбор действия
2. Выбрать таблицу. Например, все таблицы. В результате будут выведены состояния всех таблиц и появится окно для выбора следующего действия (рис. 26).



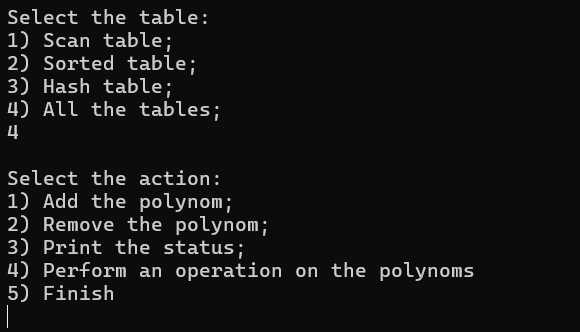
1. Выбор таблицы, которую необходимо вывести
2. Выбрать действие. Например, удалить полином из таблицы. В результате появится окно для ввода полинома, который необходимо удалить из таблицы (рис. 27).



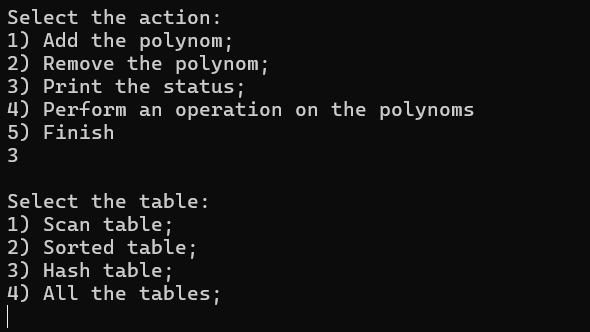
1. Выбор действия
2. Ввести полином. В результате появится окно для выбора таблицы, из которой необходимо удалить полином (рис. 28).



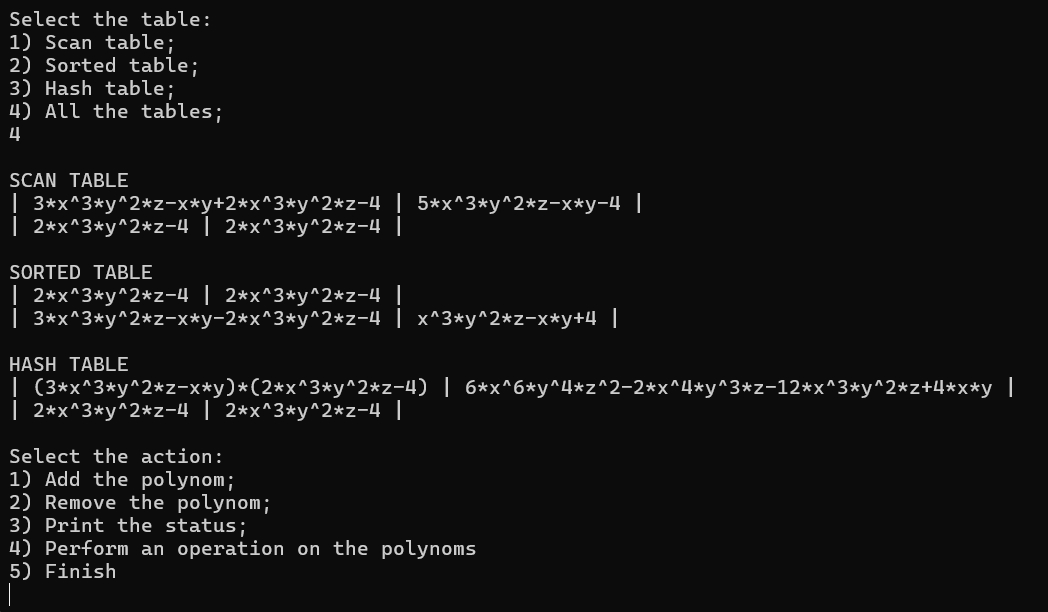
1. Ввод полинома, который необходимо удалить из таблицы
2. Выбрать таблицу. Например, все таблицы. В результате полином будет удален из всех таблиц и появится окно для выбора следующего действия (рис. 29).



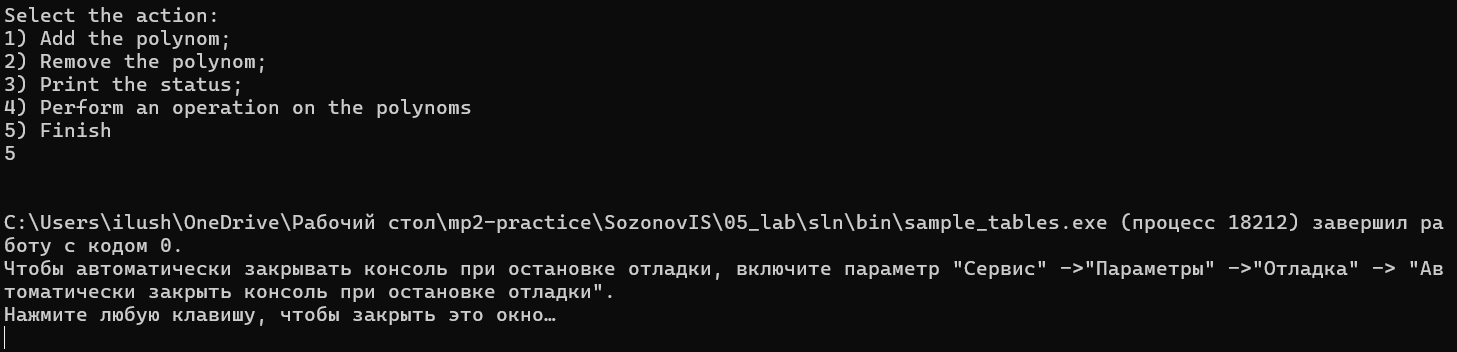
1. Выбора таблицы, из которой необходимо удалить полином
2. Выбрать действие. Например, вывести состояние таблиц. В результате появится окно для выбора таблицы, которую необходимо вывести (рис. 30).



1. Выбор действия
2. Выбрать таблицу. Например, все таблицы. В результате будут выведены состояния всех таблиц и появится окно для выбора следующего действия (рис. 31).



1. Выбор таблицы, которую необходимо вывести
2. Выбрать действие. Например, завершить. В результате приложение завершит свою работу (рис. 32).



1. Выбор дейстия

# Руководство программиста

## Использованные алгоритмы

### Просматриваемые таблицы

**Просматриваемая таблица** – это таблица, порядок расположения записей в которой произволен по отношению к значениям ключей.

|  |  |
| --- | --- |
| **Слово** | **Перевод** |
| One | 1 |
| Two | 2 |
| Three | 3 |
| Four | 4 |
| Five | 5 |

Для работы с просматриваемыми таблицами необходимо реализовать три основные операции:

* **поиск записи по ключу**

Простейшим способом поиска записи по ключу в просматриваемой таблице является метод полного просмотра (сканирования), когда искомый ключ сравнивается по очереди со всеми ключами таблицы, начиная с первого, вплоть до отыскания совпадающей записи или до исчерпания записей.

|  |  |
| --- | --- |
| **Слово** | **Перевод** |
| One | 1 |
| Two | 2 |
| Three | 3 |
| Four | 4 |
| Five | 5 |

* **вставка новой записи**

Операция вставки в просматриваемую таблицу реализуется путём добавления новой записи в конец таблицы с корректировкой номера последней занятой строки.

|  |  |
| --- | --- |
| **Слово** | **Перевод** |
| One | 1 |
| Two | 2 |
| Three | 3 |
| Four | 4 |
| Five | 5 |
| Six | 6 |

* **удаление записи**

Операция удаления записи из просматриваемой таблицы реализуется при помощи переписывания последней записи таблицы на место удаляемой и соответствующей корректировки номера последней строки.

|  |  |
| --- | --- |
| **Слово** | **Перевод** |
| One | 1 |
| Two | 2 |
| Five | 5 |
| Four | 4 |
| Six | 6 |

### Упорядоченные таблицы

**Упорядоченная таблица –** это таблица, записи в которой расположены в порядке «возрастания/убывания» ключей.

|  |  |
| --- | --- |
| **Слово** | **Перевод** |
| Five | 5 |
| Four | 4 |
| One | 1 |
| Three | 3 |
| Two | 2 |

Для упорядочивания записей в таблице наиболее эффективно использовать алгоритм быстрой сортировки:

1. Выбрать средний элемент из массива. Назовём его опорным.
2. Разбиение: перераспределение элементов в массиве таким образом, что элементы, меньшие опорного, помещаются перед ним, а большие или равные – после.
3. Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один элемент или отсутствуют элементы.

Для работы с упорядоченными таблицами необходимо реализовать три основные операции:

* **поиск записи по ключу**

Простейшим способом поиска записи по ключу в просматриваемой таблице является метод бинарного (двоичного) поиска:

1. Установить начальный индекс: индекс первой записи в таблице.
2. Установить конечный индекс: индекс последней записи в таблице.
3. Пока начальный индекс не превышает конечный индекс, делать следующее:
4. Вычислить индекс середины диапазона (например, среднее арифметическое между начальным и конечным индексом).
5. Сравнить значение ключа в записи с индексом середины с искомым значением ключа:

* Если значения равны, то вернуть эту запись.
* Если значение ключа в записи меньше искомого значения ключа, обновить начальный индекс на индекс середины + 1.

1. Если значение ключа в записи больше искомого значения ключа, обновить конечный индекс на индекс середины - 1.
2. Если искомая запись не найдена после окончания цикла, вернуть nullptr.

|  |  |
| --- | --- |
| **Слово** | **Перевод** |
| Five | 5 |
| Four | 4 |
| One | 1 |
| Three | 3 |
| Two | 2 |

* **вставка новой записи**

В упорядоченных таблицах усложняется реализация операции вставки записи, при выполнении которой для сохранения упорядоченности становится необходимой перепаковка записей таблицы:

1. Начать сравнивать новую запись с элементами таблицы, начиная с первого элемента.
2. Пока не будет найден элемент, который больше новой записи или пока не будет достигнут конец таблицы, продолжать сравнивать.
3. Если найден элемент, который больше новой записи, вставить новую запись перед данным элементом.
4. Если достигнут конец таблицы и новая запись оказывается больше всех элементов, вставить её в конец таблицы.
5. После вставки записи переопределить индексы таблицы, чтобы сохранить упорядоченность.

|  |  |
| --- | --- |
| **Слово** | **Перевод** |
| Five | 5 |
| Four | 4 |
| One | 1 |
| Six | 6 |
| Three | 3 |
| Two | 2 |

* **удаление записи**

В упорядоченных таблицах усложняется реализация операции удаления записи, при выполнении которой для сохранения упорядоченности становится необходимой перепаковка записей таблицы:

1. Начинаем сравнивать элементы таблицы с удаляемым элементом, начиная с первого элемента.
2. Если текущий элемент таблицы равен удаляемому элементу, то удаляем его из таблицы.
3. Сдвигаем все элементы таблицы справа от удаленного элемента на одну позицию влево, чтобы не было пустых ячеек.
4. Повторяем шаги 3-4 для всех остальных элементов таблицы, до тех пор, пока не пройдем все элементы.
5. Если удаляемый элемент не найден в таблице, выводим сообщение об ошибке или предупреждение.

|  |  |
| --- | --- |
| **Слово** | **Перевод** |
| Five | 5 |
| Four | 4 |
| Six | 6 |
| Three | 3 |
| Two | 2 |

### Хеш-таблицы

Обозначим через 𝐾 множество всех возможных значений ключей. Обозначим через 𝑀 число строк, которые могут быть выделены под таблицу

**Прямой доступ** по ключу возможен только при условии, когда 𝐾 ≤ 𝑀

Необходимо каждому ключу 𝑘 поставить в соответствие индекс строки 𝑖𝑛𝑑, т.е. определить на множестве ключей функцию

Функция, отображающая значение ключа в индекс строки, называется **функцией хеширования** (расстановки, перемешивания).

Таблицы, в которых данные размещены на основе хеш-функции, называются **хеш-таблицами** (таблицами с вычислимыми адресами).

Хеш-функции могу быть построены разными способами, например основанные на делении , где 𝑘 – либо значение ключа, либо сведение ключа к целому числу (например, сумма кодов символов строки).

|  |  |
| --- | --- |
| **Слово** | **Перевод** |
| Five | 5 |
|  |  |
|  |  |
|  |  |
| One | 1 |
| Four | 4 |
| Two | 2 |
|  |  |
|  |  |
| Three | 3 |
|  |  |

При условии 𝐾≫𝑀 отображение ℎ не является взаимно-однозначным, т.е.

Ситуация, когда для двух разных значений ключей и значения хеш-функции равны, т.е. записи с ключами и должны быть размещены в одной и той же строке таблицы, называется **коллизией**.

Для разрешения коллизии применяется **метод открытого перемешивания**. Он состоит в добавлении к вычисленному занятому номеру некоторого фиксированного смещения (повторное перемешивание) ; если новый адрес hh(k) также является занятым, следует повторить процедуру повторного перемешивания до тех пор, пока не обнаружится свободная строка, либо таблица не будет исчерпана (если значения p и M являются взаимнопростыми, открытое перемешивание обеспечивает нахождение свободной строки массива).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  | 1 |  |  | 1 |  |  | 1 |  |  | 1 | Five |
| 2 |  |  | 2 |  |  | 2 |  |  | 2 |  |  | 2 |  |
| 3 |  |  | 3 |  |  | 3 |  |  | 3 |  |  | 3 |  |
| 4 | One |  | 4 | One |  | 4 | One |  | 4 | One |  | 4 | One |
| 5 |  |  | 5 |  |  | 5 |  |  | 5 | Four |  | 5 | Four |
| 6 |  |  | 6 | Two |  | 6 | Two |  | 6 | Two |  | 6 | Two |
| 7 |  |  | 7 |  |  | 7 |  |  | 7 |  |  | 7 |  |
| 8 |  |  | 8 |  |  | 8 |  |  | 8 |  |  | 8 |  |
| 9 |  |  | 9 |  |  | 9 | Three |  | 9 | Three |  | 9 | Three |
| 10 |  |  | 10 |  |  | 10 |  |  | 10 |  |  | 10 |  |

Для работы с хеш-таблицами необходимо реализовать три основные операции:

* **поиск записи по ключу**

1. Передать искомый ключ в хеш-функцию.
2. Если по полученному индексу от хеш-функции в последовательности записей лежит запись с искомым ключом, то завершить поиск.
3. Иначе возникла коллизия и необходимо получить новый индекс, сделав шаг.
4. Проверяем таким образом каждый новый индекс, пока не найдем запись с ключом, соответствующим искомому, либо пока не просмотрим все элементы таблицы или не дойдём до индекса, у которого нет записи.

|  |  |
| --- | --- |
| **Слово** | **Перевод** |
| Five | 5 |
|  |  |
|  |  |
|  |  |
| One | 1 |
| Four | 4 |
| Two | 2 |
|  |  |
|  |  |
| Three | 3 |
|  |  |

* **вставка новой записи**

1. Если таблица полная, то завершить вставку.
2. Выполнить поиск по ключу записи, которую необходимо вставить.
3. Если запись с таким ключом уже есть, то завершить вставку с ошибкой (нарушение принципа уникальности ключей).
4. Создать по заданному ключу и значению новую запись.
5. Вставить её по индексу свободного места, найденного при выполнении поиска.

|  |  |
| --- | --- |
| **Слово** | **Перевод** |
| Five | 5 |
|  |  |
| Six | 6 |
|  |  |
| One | 1 |
| Four | 4 |
| Two | 2 |
|  |  |
|  |  |
| Three | 3 |
|  |  |

* **удаление записи**

1. Если таблица не имеет записей, то завершаем удаление.
2. Выполняем поиск записи по ключу, запись с которым необходимо удалить.
3. Если записи с этим ключом нет, завершаем удаление.
4. Получаем от операции поиска индекс искомой записи.
5. Удаляем запись и на её место добавляем фиктивные значения, для указания того факта, что по этому индексу был удалён элемент.

|  |  |
| --- | --- |
| **Слово** | **Перевод** |
| Five | 5 |
|  |  |
| Six | 6 |
|  |  |
| Four | 4 |
| Two | 2 |
|  |  |
|  |  |
| Three | 3 |
|  |  |

## Описание классов

### Схема наследования классов

**TabRecord**

**Table**

**ScanTable**

**SortedTable**

**HashTable**

**ArrayHashTable**

1. Схема наследования классов

### Структура TabRecord

Объявление структуры:

template <typename TKey, typename TData>

struct TabRecord {

TKey key;

TData\* data;

TabRecord();

TabRecord(TKey \_key, TData \*\_data);

friend ostream& operator<<(ostream& out, const TabRecord<TKey, TData>& record;

};

Поля:

key – ключ записи.

data – указатель на значение записи.

Конструкторы:

TabRecord();

Назначение: установка значений полей класса TabRecord по умолчанию.

Входные данные: отсутствуют.

Выходные данные: отсутствуют.

TabRecord(TKey \_key, TData \*\_data));

Назначение: инициализация полей класса TabRecord.

Входные данные: \_key – ключ записи, \_data – указатель на значение записи.

Выходные данные: отсутствуют.

Методы:

friend ostream& operator<<(ostream& out, const TabRecord<TKey, TData>& record);

Назначение: вывод записи.

Входные данные: out – ссылка на стандартный поток вывода, record – константная ссылка на выводимую запись.

Выходные данные: ссылка на поток вывода.

### Класс Table

Объявление класса:

template <typename TKey, typename TData>

class Table {

protected:

int count;

int maxSize;

int currPos;

public:

Table(int max\_size);

virtual TabRecord<TKey, TData>\* GetCurrent()const = 0;

virtual TabRecord<TKey, TData>\* Search(TKey \_key) = 0;

virtual void Insert(TKey \_key, TData\* \_data) = 0;

virtual void Remove(TKey \_key) = 0;

virtual bool IsFull()const noexcept;

virtual bool IsEmpty()const noexcept;

virtual bool IsEnded()const noexcept;

virtual void Reset() noexcept;

virtual void Next();

};

Поля:

count – количество записей в таблице.

maxSize – размер таблицы (максимально количество записей).

currPos – индекс текущей записи.

Конструкторы:

Table(int max\_size);

Назначение: инициализация поля maxSize класса Table.

Входные данные: max\_size – размер таблицы (максимально количество записей).

Выходные данные: отсутствуют.

Методы:

virtual TabRecord<TKey, TData>\* GetCurrent() const = 0;

Назначение: получение текущей записи.

Входные данные: отсутствуют.

Выходные данные: указатель на текущую запись.

virtual TabRecord<TKey, TData>\* Search(TKey \_key) = 0;

Назначение: поиск записи по ключу.

Входные данные: \_key – ключ искомой записи.

Выходные данные: указатель на найденную запись.

virtual void Insert(TKey \_key, TData\* \_data) = 0;

Назначение: вставка новой записи в таблицу.

Входные данные: \_key – ключ добавляемой записи, \_data – указатель на значение добавляемой записи.

Выходные данные: отсутствуют.

virtual void Remove(TKey \_key) = 0;

Назначение: удаление записи по ключу.

Входные данные: \_key – ключ удаляемой записи.

Выходные данные: отсутствуют.

virtual bool IsFull() const noexcept;

Назначение: проверка, является ли таблица заполненной.

Входные данные: отсутствуют.

Выходные данные: результат проверки (true – заполненная, false – не заполненная).

virtual bool IsEmpty() const noexcept;

Назначение: проверка, является ли таблица пустой.

Входные данные: отсутствуют.

Выходные данные: результат проверки (true – пустая, false – не пустая).

virtual bool IsEnded() const noexcept;

Назначение: проверка, достигли ли конца таблицы.

Входные данные: отсутствуют.

Выходные данные: результат проверки (true – достигли, false – не достигли).

virtual void Reset() noexcept;

Назначение: переход в первой записи таблицы.

Входные данные: отсутствуют.

Выходные данные: отсутствуют.

virtual void Next();

Назначение: переход к следующей записи таблицы.

Входные данные: отсутствуют.

Выходные данные: отсутствуют.

### Класс ScanTable

Объявление класса:

template <typename TKey, typename TData>

class ScanTable : public Table<TKey, TData> {

protected:

TabRecord<TKey, TData>\*\* recs;

public:

ScanTable(int max\_size);

ScanTable(const ScanTable<TKey, TData>& table);

~ScanTable();

TabRecord<TKey, TData>\* GetCurrent() const override;

TabRecord<TKey, TData>\* Search(TKey \_key) override;

void Insert(TKey \_key, TData\* \_data) override;

void Remove(TKey \_key) override;

friend ostream& operator<<(ostream& out, const ScanTable<TKey, TData>& table);

};

Поля:

recs – массив указателей на запись.

Конструкторы:

ScanTable(int max\_size);

Назначение: инициализация поля maxSize класса ScanTable.

Входные данные: max\_size – размер таблицы (максимально количество записей).

Выходные данные: отсутствуют.

ScanTable(const ScanTable<TKey, TData>& table);

Назначение: создание копии просматриваемой таблицы.

Входные данные: table – константная ссылка на копируемую просматриваемую таблицу.

Выходные данные: отсутствуют.

Деструктор:

~ScanTable();

Назначение: освобождение памяти, занимаемой динамическими полями класса ScanTable.

Входные данные: отсутствуют.

Выходные данные: отсутствуют.

Методы:

TabRecord<TKey, TData>\* GetCurrent() const override;

Назначение: получение текущей записи.

Входные данные: отсутствуют.

Выходные данные: указатель на текущую запись.

TabRecord<TKey, TData>\* Search(TKey \_key) override;

Назначение: поиск записи по ключу.

Входные данные: \_key – ключ искомой записи.

Выходные данные: указатель на найденную запись.

void Insert(TKey \_key, TData\* \_data) override;

Назначение: вставка новой записи в таблицу.

Входные данные: \_key – ключ добавляемой записи, \_data – указатель на значение добавляемой записи.

Выходные данные: отсутствуют.

void Remove(TKey \_key) override;

Назначение: удаление записи по ключу.

Входные данные: \_key – ключ удаляемой записи.

Выходные данные: отсутствуют.

friend ostream& operator<<(ostream& out, const ScanTable<TKey, TData>& table);

Назначение: вывод просматриваемой таблицы.

Входные данные: out – ссылка на стандартный поток вывода, table – константная ссылка на выводимую таблицу.

Выходные данные: ссылка на поток вывода.

### Класс SortedTable

Объявление класса:

template <typename TKey, typename TData>

class SortedTable : public ScanTable<TKey, TData> {

private:

void QuickSort(TabRecord<TKey, TData>\*\* records, int n);

public:

SortedTable(int max\_size);

SortedTable(const ScanTable<TKey, TData>& table);

TabRecord<TKey, TData>\* GetCurrent() const override;

TabRecord<TKey, TData>\* Search(TKey \_key) override;

void Insert(TKey \_key, TData\* \_data) override;

void Remove(TKey \_key) override;

};

Конструкторы:

SortedTable(int max\_size);

Назначение: инициализация поля maxSize класса SortedTable.

Входные данные: max\_size – размер таблицы (максимально количество записей).

Выходные данные: отсутствуют.

SortedTable(const ScanTable<TKey, TData>& table);

Назначение: создание копии упорядоченной таблицы.

Входные данные: table – константная ссылка на копируемую просматриваемую таблицу.

Выходные данные: отсутствуют.

Методы:

void QuickSort(TabRecord<TKey, TData>\*\* records, int n);

Назначение: упорядочивание записей таблицы.

Входные данные: records – массив указателей на запись, n – количество записей.

Выходные данные: отсутствуют

TabRecord<TKey, TData>\* GetCurrent() const override;

Назначение: получение текущей записи.

Входные данные: отсутствуют.

Выходные данные: указатель на текущую запись.

TabRecord<TKey, TData>\* Search(TKey \_key) override;

Назначение: поиск записи по ключу.

Входные данные: \_key – ключ искомой записи.

Выходные данные: указатель на найденную запись.

void Insert(TKey \_key, TData\* \_data) override;

Назначение: вставка новой записи в таблицу.

Входные данные: \_key – ключ добавляемой записи, \_data – указатель на значение добавляемой записи.

Выходные данные: отсутствуют.

void Remove(TKey \_key) override;

Назначение: удаление записи по ключу.

Входные данные: \_key – ключ удаляемой записи.

Выходные данные: отсутствуют.

### Класс HashTable

Объявление класса:

template <typename TKey, typename TData>

class HashTable : public Table<TKey, TData> {

protected:

int hashFunc(const TKey \_key);

public:

HashTable(int max\_size);

};

Конструкторы:

HashTable(int max\_size);

Назначение: инициализация поля maxSize класса HashTable.

Входные данные: max\_size – размер таблицы (максимально количество записей).

Выходные данные: отсутствуют.

Методы:

int hashFunc(const TKey \_key);

Назначение: вычисление функции хэширования.

Входные данные: \_key – ключ записи.

Выходные данные: первый возможный индекс записи в таблице.

### Класс ArrayHashTable

Объявление класса:

template <typename TKey, typename TData>

class ArrayHashTable : public HashTable<TKey, TData> {

private:

TabRecord<TKey, TData>\*\* recs;

TabRecord<TKey, TData>\* pMark;

int freePos;

int hashStep;

int GetNextPos(int hashValue) const;

public:

ArrayHashTable(int max\_size, int step);

ArrayHashTable(const ArrayHashTable<TKey, TData>& table);

~ArrayHashTable();

TabRecord<TKey, TData>\* GetCurrent() const;

TabRecord<TKey, TData>\* Search(TKey \_key);

void Insert(TKey \_key, TData\* \_data);

void Remove(TKey \_key);

void Reset() noexcept;

void Next();

friend ostream& operator<<(ostream& out, const ArrayHashTable<TKey, TData>& table);

}

Поля:

recs – массив указателей на запись.

pMark – указатель на запись, означающую, что она была удалена из таблицы.

freePos – индекс пустой строки хеш-таблицы.

hashStep – линейный шаг в методе «открытое перемешивание».

Конструкторы:

ArrayHashTable(int max\_size, int step);

Назначение: инициализация полей maxSize и hashStep класса ArrayHashTable.

Входные данные: max\_size – размер таблицы (максимально количество записей),

hashStep – линейный шаг в методе «открытое перемешивание».

Выходные данные: отсутствуют.

ArrayHashTable(const ArrayHashTable<TKey, TData>& table);

Назначение: создание копии хеш-таблицы.

Входные данные: table – константная ссылка на копируемую хэш-таблицу.

Выходные данные: отсутствуют.

Деструктор:

~ArrayHashTable();

Назначение: освобождение памяти, занимаемой динамическими полями класса ArrayHashTable.

Входные данные: отсутствуют.

Выходные данные: отсутствуют.

Методы:

int GetNextPos(int hashValue) const;;

Назначение: получение индекса следующей записи.

Входные данные: hashValue – текущий индекс записи в хэш-таблице.

Выходные данные: индекс следующей записи.

TabRecord<TKey, TData>\* GetCurrent() const override;

Назначение: получение текущей записи.

Входные данные: отсутствуют.

Выходные данные: указатель на текущую запись.

TabRecord<TKey, TData>\* Search(TKey \_key) override;

Назначение: поиск записи по ключу.

Входные данные: \_key – ключ искомой записи.

Выходные данные: указатель на найденную запись.

void Insert(TKey \_key, TData\* \_data) override;

Назначение: вставка новой записи в таблицу.

Входные данные: \_key – ключ добавляемой записи, \_data – указатель на значение добавляемой записи.

Выходные данные: отсутствуют.

void Remove(TKey \_key) override;

Назначение: удаление записи по ключу.

Входные данные: \_key – ключ удаляемой записи.

Выходные данные: отсутствуют.

void Reset() noexcept;

Назначение: переход в первой записи таблицы.

Входные данные: отсутствуют.

Выходные данные: отсутствуют.

void Next();

Назначение: переход к следующей записи таблицы.

Входные данные: отсутствуют.

Выходные данные: отсутствуют.

friend ostream& operator<<(ostream& out, const ArrayHashTable<TKey, TData>& table);

Назначение: вывод хеш-таблицы.

Входные данные: out – ссылка на стандартный поток вывода, table – константная ссылка на выводимую хеш-таблицу.

Выходные данные: ссылка на поток вывода.

# Заключение

В ходе лабораторной работы были изучены основные термины и понятия, связанные с таблицами, а также наиболее эффективные способы их представления (хранения).

На основе подготовленной теоретической базы, были реализованы структура для представления записей таблицы TabRecord и классы для представления просматриваемых таблиц ScanTable, упорядоченных таблиц SortedTable, хэш-таблиц HashTable и хэш-таблиц, основанных на методе открытого перемешивания ArrayHashTable со всеми необходимыми операциями над таблицами. Для проверки работоспособности и эффективности реализации перечисленных выше классов были написано приложение sample\_tables, а также модульные тесты.

# Литература

1. Барышева И.В. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017 – 105 с.
2. Организация доступа по имени к структурам данных [https://studfile.net/preview/3392055/page:5]

# Приложения

## Приложение А. Реализация структуры TabRecord

template <typename TKey, typename TData>

TabRecord<TKey, TData>::TabRecord() {

key = TKey();

data = nullptr;

}

template <typename TKey, typename TData>

TabRecord<TKey, TData>::TabRecord(TKey \_key, TData\* \_data) {

key = \_key;

data = new TData(\*\_data);

}

## Приложение Б. Реализация класса Table

template <typename TKey, typename TData>

Table<TKey, TData>::Table(int max\_size) {

if (max\_size <= 0)

throw exception("negative or zero max size");

count = 0;

maxSize = max\_size;

currPos = -1;

}

template <typename TKey, typename TData>

bool Table<TKey, TData>::IsFull()const noexcept {

return count == maxSize;

}

template <typename TKey, typename TData>

bool Table<TKey, TData>::IsEmpty()const noexcept {

return count == 0;

}

template <typename TKey, typename TData>

bool Table<TKey, TData>::IsEnded()const noexcept {

return currPos == maxSize;

}

template <typename TKey, typename TData>

void Table<TKey, TData>::Reset() noexcept {

currPos = 0;

}

template <typename TKey, typename TData>

void Table<TKey, TData>::Next() {

if (IsEnded())

throw exception("end of the table");

currPos++;

}

## Приложение В. Реализация класса ScanTable

template <typename TKey, typename TData>

ScanTable<TKey, TData>::ScanTable(int max\_size) : Table<TKey, TData>(max\_size) {

recs = new TabRecord<TKey, TData>\*[max\_size];

}

template <typename TKey, typename TData>

ScanTable<TKey, TData>::ScanTable(const ScanTable<TKey, TData>& table) : Table<TKey, TData>(table.maxSize) {

recs = new TabRecord<TData, TKey>\*[maxSize];

for (int i = 0; i < maxSize; i++) {

recs[i] = new TabRecord<TData, TKey>(table.recs[i]->key, table.recs[i]->data);

}

}

template <typename TKey, typename TData>

ScanTable<TKey, TData>::~ScanTable() {

if (!IsEmpty()) {

for (int i = 0; i < count; i++)

delete recs[i];

}

delete[] recs;

}

template <typename TKey, typename TData>

TabRecord<TKey, TData>\* ScanTable<TKey, TData>::GetCurrent() const {

if (IsEmpty())

throw exception("table is empty");

return recs[currPos];

}

template <typename TKey, typename TData>

TabRecord<TKey, TData>\* ScanTable<TKey, TData>::Search(TKey \_key) {

if (IsEmpty())

return nullptr;

for (int i = 0; i < count; i++)

if (recs[i]->key == \_key) {

currPos = i;

return recs[i];

}

return nullptr;

}

template <typename TKey, typename TData>

void ScanTable<TKey, TData>::Insert(TKey \_key, TData\* \_data) {

if (IsFull())

throw exception("table is full");

if (Search(\_key) != nullptr)

throw exception("record with this key already existed");

recs[count++] = new TabRecord<TKey, TData>(\_key, \_data);

}

template <typename TKey, typename TData>

void ScanTable<TKey, TData>::Remove(TKey \_key) {

if (IsEmpty())

throw exception("table is empty");

if (Search(\_key) == nullptr)

throw exception("no such records");

delete recs[currPos];

recs[currPos] = recs[--count];

}

## Приложение Г. Реализация класса SortedTable

template <typename TKey, typename TData>

void SortedTable<TKey, TData>::QuickSort(TabRecord<TKey, TData>\*\* records, int n) {

int i = 0;

int j = n - 1;

TabRecord<TKey, TData>\* mid = records[(i + j) / 2];

do {

while (records[i]->key < mid->key)

i++;

while (records[j]->key > mid->key)

j--;

if (i <= j) {

TabRecord<TKey, TData>\* tmp = records[i];

records[i] = records[j];

records[j] = tmp;

i++;

j--;

}

} while (i <= j);

if (j > 0)

QuickSort(records, j + 1);

if (i < n)

QuickSort(records, n - 1);

}

template <typename TKey, typename TData>

SortedTable<TKey, TData>::SortedTable(int max\_size) : ScanTable<TKey, TData>(max\_size) {}

template <typename TKey, typename TData>

SortedTable<TKey, TData>::SortedTable(const ScanTable<TKey, TData>& table) : ScanTable<TKey, TData>(table) {

if (IsEmpty())

return;

this->QuickSort(this->recs, this->count);

}

template <typename TKey, typename TData>

TabRecord<TKey, TData>\* SortedTable<TKey, TData>::GetCurrent()const {

if (IsEmpty())

throw exception("table is empty");

return recs[currPos];

}

template <typename TKey, typename TData>

TabRecord<TKey, TData>\* SortedTable<TKey, TData>::Search(TKey \_key) {

int left = 0, right = count - 1;

TabRecord<TKey, TData>\* record = nullptr;

while (left <= right) {

int mid = (left + right) / 2;

if (recs[mid]->key == \_key) {

left = mid + 1;

right = mid;

record = recs[mid];

}

else if (recs[mid]->key > \_key)

right = mid - 1;

else

left = mid + 1;

}

currPos = right;

return record;

}

template <typename TKey, typename TData>

void SortedTable<TKey, TData>::Insert(TKey \_key, TData\* \_data) {

if (IsFull())

throw exception("table is full");

if (Search(\_key) != nullptr)

throw exception("record with this key already existed");

for (int i = count - 1; i > currPos; i--) {

recs[i + 1] = recs[i];

}

recs[++currPos] = new TabRecord<TKey, TData>(\_key, \_data);

count++;

}

template <typename TKey, typename TData>

void SortedTable<TKey, TData>::Remove(TKey \_key) {

if (IsEmpty())

throw exception("table is empty");

if (Search(\_key) == nullptr)

throw exception("no such records");

delete recs[currPos];

for (int i = currPos; i < count - 1; i++)

recs[i] = recs[i + 1];

count--;

}

## Приложение Д. Реализация класса HashTable

template <typename TKey, typename TData>

int HashTable<TKey, TData>::hashFunc(const TKey \_key) {

return std::hash<TKey>{}(\_key) % maxSize;

}

template <typename TKey, typename TData>

HashTable<TKey, TData>::HashTable(int max\_size) : Table<TKey, TData>(max\_size) {}

## Приложение Е. Реализация класса ArrayHashTable

template <typename TKey, typename TData>

int ArrayHashTable<TKey, TData>::GetNextPos(int hashValue) const {

return (hashValue + hashStep) % maxSize;

}

template <typename TKey, typename TData>

ArrayHashTable<TKey, TData>::ArrayHashTable(int max\_size, int step) : HashTable<TKey, TData>(max\_size) {

count = 0;

maxSize = max\_size;

currPos = 0;

recs = new TabRecord<TKey, TData>\*[max\_size];

for (int i = 0; i < max\_size; i++)

recs[i] = nullptr;

pMark = new TabRecord<TKey, TData>();

freePos = -1;

hashStep = step;

}

template <typename TKey, typename TData>

ArrayHashTable<TKey, TData>::ArrayHashTable(const ArrayHashTable<TKey, TData>& table) :

ArrayHashTable<TKey, TData>(table.maxSize, table.hashStep) {

freePos = table.freePos;

hashStep = table.hashStep;

recs = new TabRecord<TKey, TData>\*[maxSize];

pMark = new TabRecord<TKey, TData>();

for (int i = 0; i < maxSize; i++) {

if (table.recs[i] == nullptr)

recs[i] = nullptr

else if (table.recs[i] == table.pMark)

recs[i] = pMark;

else

recs[i] = new TabRecord<TKey, TData>(table.recs[i]->key, table.recs[i]->data);

}

}

template <typename TKey, typename TData>

ArrayHashTable<TKey, TData>::~ArrayHashTable() {

for (int i = 0; i < maxSize; i++) {

if (recs[i] == pMark)

recs[i] = nullptr;

else if (recs[i] != nullptr)

delete recs[i];

}

delete[] recs;

delete pMark;

}

template <typename TKey, typename TData>

TabRecord<TKey, TData>\* ArrayHashTable<TKey, TData>::GetCurrent() const {

if (IsEmpty())

throw exception("table is empty");

return recs[currPos];

}

template <typename TKey, typename TData>

TabRecord<TKey, TData>\* ArrayHashTable<TKey, TData>::Search(TKey \_key) {

TabRecord<TKey, TData>\* res = nullptr;

currPos = hashFunc(\_key);

freePos = -1;

for (int i = 0; i < count; i++) {

if (recs[currPos] == nullptr) {

if (freePos == -1)

freePos = currPos;

break;

}

else if (recs[currPos] == pMark) {

if (freePos == -1)

freePos = currPos;

}

else if (recs[currPos]->key == \_key) {

res = recs[currPos];

break;

}

currPos = GetNextPos(currPos);

}

return res;

}

template <typename TKey, typename TData>

void ArrayHashTable<TKey, TData>::Insert(TKey \_key, TData\* \_data) {

if (IsFull())

throw exception("table is full");

if (Search(\_key) != nullptr)

throw exception("record with this key already existed");

if (freePos != -1)

currPos = freePos;

recs[currPos] = new TabRecord<TKey, TData>(\_key, \_data);

count++;

}

template <typename TKey, typename TData>

void ArrayHashTable<TKey, TData>::Remove(TKey \_key) {

if (IsEmpty())

throw exception("table is empty");

if (Search(\_key) == nullptr)

throw exception("no such records");

delete recs[currPos];

recs[currPos] = pMark;

count--;

}

template <typename TKey, typename TData>

void ArrayHashTable<TKey, TData>::Reset() noexcept {

currPos = 0;

while (!IsEnded()) {

if (recs[currPos] == nullptr && recs[currPos] == pMark)

currPos++;

}

}

template <typename TKey, typename TData>

void ArrayHashTable<TKey, TData>::Next() {

if (IsEnded())

throw exception("end of the table");

currPos++;

while (!IsEnded()) {

if (recs[currPos] == nullptr && recs[currPos] == pMark)

currPos++;

}

}