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:

«Реализация и использование различных структур таблиц для хранения и обработки полиномов»

Выполнил(а): студент(ка) группы 3822Б1ФИ2
_____ / Савченко М.П./
Подпись

Проверил: к.т.н, доцент каф. ВВиСП
_____ / Кустикова В.Д./
Подпись

Нижний Новгород
2024

Содержание

Введение

В предыдущей лабораторной работе была разработана и реализована структура данных для представления и работы с полиномами. Это включало создание классов для мономов (**`TMonom`**) и полиномов (**`TPolynom`**), которые позволяли эффективно хранить и манипулировать полиномами с несколькими переменными. Эти структуры данных обеспечили базовую функциональность для работы с полиномами, включая операции сложения, вычитания, умножения и дифференцирования.

В данной лабораторной работе основное внимание уделяется разработке и реализации различных структур таблиц, которые будут использоваться для хранения и обработки полиномов. В частности, реализуются следующие типы таблиц:

1. **`ScanTable`** - таблица с последовательным (линейным) поиском.
2. **`SortedTable`** - отсортированная таблица, поддерживающая быстрый поиск за счет упорядоченности элементов.
3. **`ArrayHashTable`** - хэш-таблица, обеспечивающая быстрый доступ к элементам с использованием хэш-функций.

Каждая из этих структур имеет свои особенности и преимущества в зависимости от типа задач и требований к производительности. В данной работе будет подробно рассмотрена реализация этих таблиц, а также приведены примеры их использования для хранения и обработки полиномов.

1 Постановка задачи

Целью данной лабораторной работы является разработка и реализация различных структур таблиц для хранения и обработки полиномов, основанных на ранее разработанных структурах данных для мономов и полиномов. В частности, необходимо реализовать и протестировать следующие классы:

1. TabRecord:
   - Структура записи таблицы, содержащая ключ и указатель на данные.
   - Реализация конструкторов, деструкторов и операторов присваивания.
2. Table:
   - Абстрактный класс таблицы с базовой функциональностью для вставки, удаления, поиска и доступа к записям.
   - Определение методов для проверки состояния таблицы (пустая, полная) и управления текущей позицией в таблице.
3. HashTable:
   - Абстрактный класс хэш-таблицы, наследуемый от `Table`.
   - Реализация базового функционала с добавлением виртуального метода `hash_func` для вычисления хэш-функции.
4. ScanTable:
   - Таблица с последовательным (линейным) поиском, наследуемая от `Table`.
   - Реализация методов вставки, удаления и поиска записей.
5. SortedTable:
   - Отсортированная таблица, наследуемая от `ScanTable`, с реализацией методов поддержания упорядоченности элементов.
   - Реализация быстрой сортировки и поиска записей.
6. ArrayHashTable<TKey, TData>:
   - Хэш-таблица, наследуемая от `HashTable`, с реализацией хэш-функции и методов для обработки коллизий.
   - Реализация методов вставки, удаления и поиска записей с использованием хэширования.

Основная задача заключается в следующем:

- Реализовать указанные структуры данных, обеспечив корректное выполнение всех необходимых операций.

- Провести тестирование каждой реализованной структуры с использованием библиотеки Google Test для проверки корректности работы методов.
- Продемонстрировать использование этих таблиц на примере хранения и обработки полиномов, представленных с помощью ранее разработанных классов `TMonom` и `TPolynom`.

Основные этапы выполнения работы:

1. Разработка классов и методов:
   - Реализовать классы `TabRecord`, `Table`, `HashTable`, `ScanTable`, `SortedTable` и `ArrayHashTable`.
   - Обеспечить корректную реализацию всех методов, включая вставку, удаление, поиск и доступ к элементам.
2. Тестирование:
   - Написать тесты для каждого класса с использованием библиотеки Google Test.
   - Проверить корректность работы всех методов, включая конструкторы и деструкторы.
3. Демонстрация работы:
   - Создать примеры использования разработанных таблиц для хранения и обработки полиномов.
   - Показать преимущества и особенности каждой структуры на примере работы с полиномами.

Результатом работы будет полная реализация и тестирование различных структур таблиц, а также их применение для эффективной обработки математических выражений, представленных в виде полиномов.

2 Руководство пользователя

2.1 Приложение для демонстрации работы таблиц полиномов

1. Запустите приложение с названием sample_table.exe. В результате появится окно, показанное ниже (рис. 1). На нем вам будет предложено выбрать меню для дальнейшей работы. Так же обозначены выбранные таблицы для работы. Для выбора введите соответствующий номер.

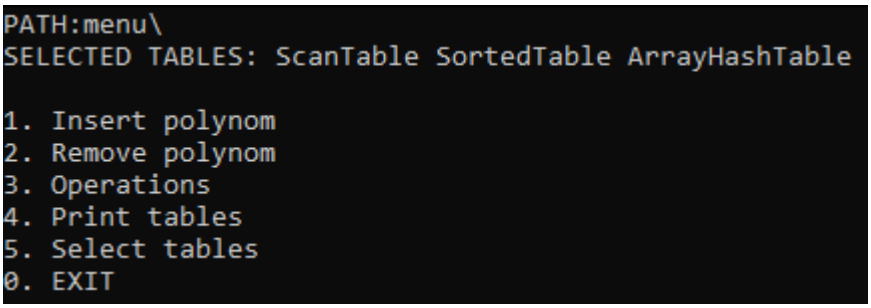


Рис. 1. Основное окно программы.

2. При выборе меню «1. Insert polynom» вам будет предложено ввести полином, для добавления его в выбранную таблицу (рис. 2). Так же будет выведено, был ли полином добавлен в таблицу или произошла ошибка (рис. 3).

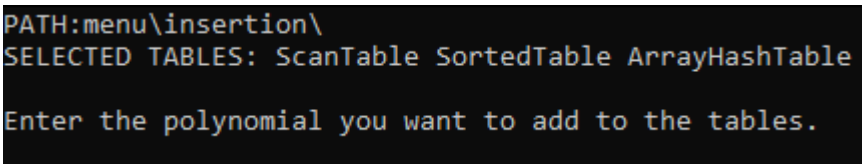


Рис. 2. Меню «1. Insert polynom».

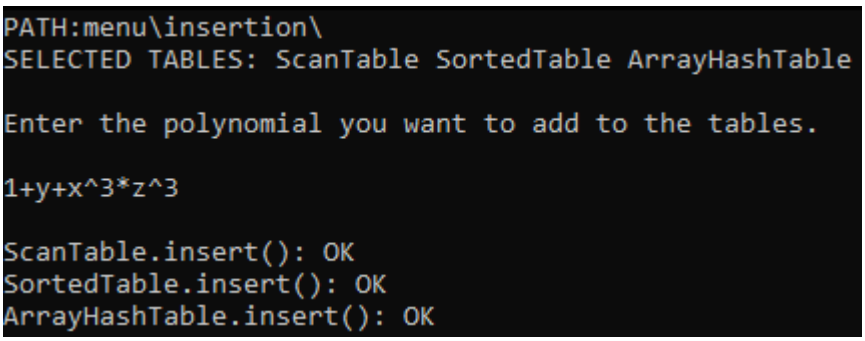


Рис. 3. Пример добавления полинома в таблицу.

3. При выборе меню «2. Remove polynom» вам будет предложено ввести полином, который будет удален из выбранных таблиц (рис. 4). Так же будет выведено, был ли полином удален из таблицы или произошла ошибка (рис. 5).

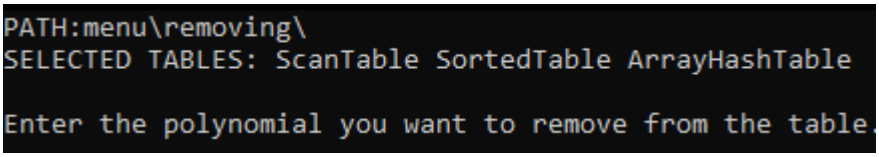


Рис. 4. Меню «2. Remove polynom».

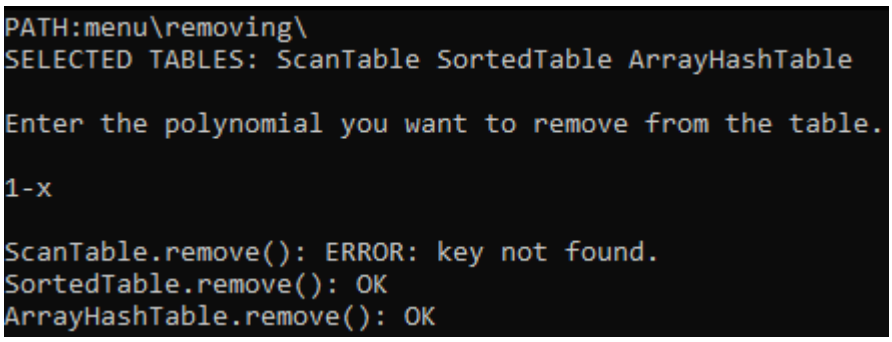


Рис. 5. Пример удаления полинома из таблиц.

4. При выборе меню «3. Operations» сначала вам будет предложено выбрать таблицу из списка, из которой будут браться полиномы (рис. 6). Затем будет предложено выбрать из списка арифметическую операцию (рис. 7). Потом вы должны ввести полиномы из таблицы, над которыми будет произведена выбранная операция (рис. 8). Если полинома в таблице не будет, то будет выведена ошибка и вас перенаправит в главное меню. В конце будет выведен результат операции, и вам будет предложено сохранить результат в данную таблицу (рис. 9).

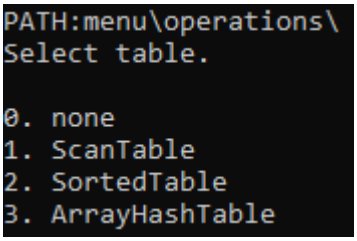


Рис. 6. Меню «3. Operations». Выбор таблицы.

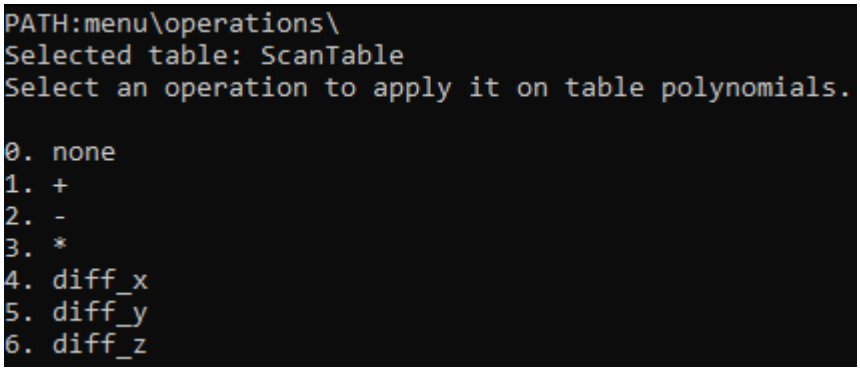


Рис. 7. Меню «3. Operations». Выбор операции.

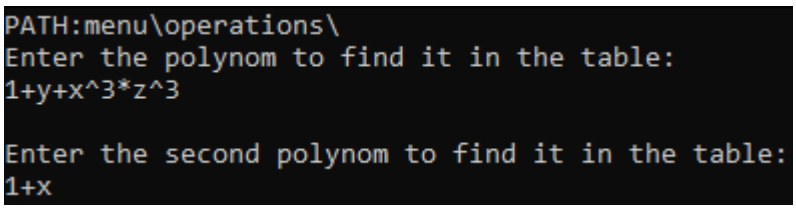


Рис. 8. Меню «3. Operations». Ввод полиномов из таблицы.

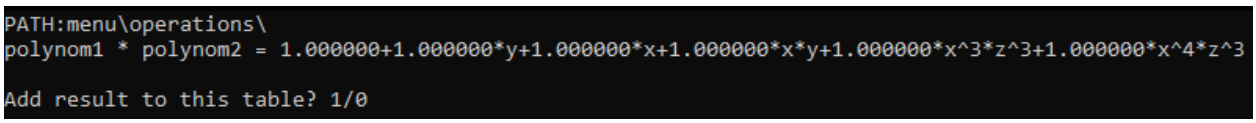


Рис. 9. Меню «3. Operations». Вывод результата операции.

5. При выборе меню «4. Print tables» будут полностью выведены выбранные таблицы (рис. 10).

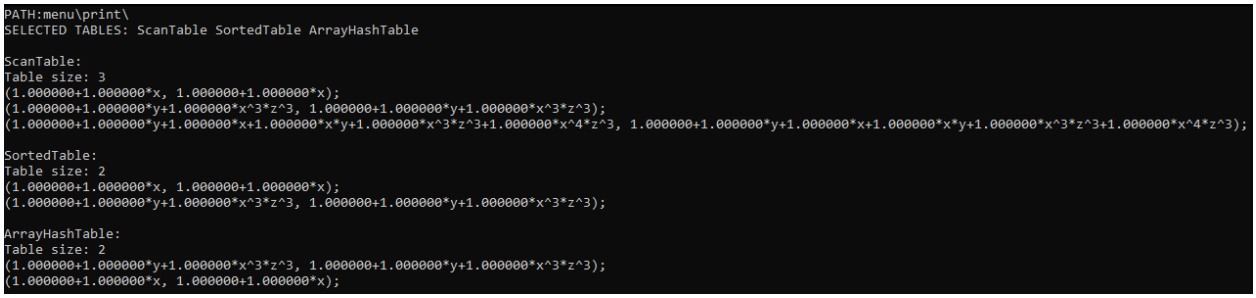


Рис. 10. Вывод выбранных таблиц.

6. При выборе меню «5. Select tables» вам будет предложено выбрать таблицы, с которыми будет производиться работа далее (рис. 11). Введите номер таблицы, чтобы выбрать ее или отменить выбор. Чтобы вернуться обратно, введите 0.

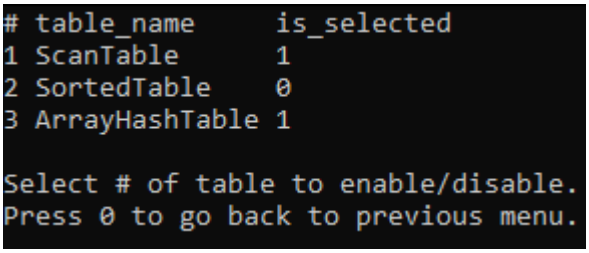


Рис. 11. Меню выбора таблиц.

2.2 Приложение для демонстрации работы полиномов

Описано в отчете к предыдущей лабораторной работе.

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Таблица линейного поиска

Алгоритм таблицы линейного поиска (также известный как таблица с последовательным поиском) основан на простом принципе последовательного перебора всех элементов таблицы для выполнения операций поиска, вставки и удаления.

1. Поиск:

- Для поиска элемента в таблице линейного поиска начинается с первого элемента.
- Каждый элемент таблицы последовательно сравнивается с ключом поиска до тех пор, пока не будет найден элемент с совпадающим ключом или пока не будут просмотрены все элементы таблицы.
- Если элемент с заданным ключом найден, возвращается указатель на этот элемент; в противном случае возвращается значение nullptr, указывающее на отсутствие элемента в таблице.

2. Вставка:

- Вставка элемента в таблицу линейного поиска также начинается с первого элемента.
- Каждый элемент таблицы последовательно сравнивается с ключом вставляемого элемента до тех пор, пока не будет найден элемент с ключом, равным или большим ключу вставляемого элемента, или пока не будет достигнут конец таблицы.
- Если элемент с таким ключом уже существует в таблице, вставка не производится; в противном случае вставляемый элемент вставляется в таблицу на место текущего элемента или в конец таблицы, если такой элемент не найден.

3. Удаление:

- Удаление элемента из таблицы линейного поиска также начинается с первого элемента.
- Каждый элемент таблицы последовательно сравнивается с ключом удаляемого элемента до тех пор, пока не будет найден элемент с совпадающим ключом или пока не будет достигнут конец таблицы.

- Если элемент с заданным ключом найден, он удаляется из таблицы путем сдвига всех последующих элементов на одну позицию влево, чтобы занять его место.

3.1.2 Отсортированная таблица

Алгоритм отсортированной таблицы с бинарным поиском применяет бинарный поиск для эффективного выполнения операций поиска, вставки и удаления элементов в отсортированной последовательности.

1. Поиск:

- Для поиска элемента в отсортированной таблице с бинарным поиском сначала определяется середина отсортированной последовательности.
- Если ключ поиска совпадает с ключом в середине последовательности, элемент найден.
- Если ключ поиска меньше ключа в середине, поиск продолжается в левой половине последовательности; в противном случае поиск продолжается в правой половине.
- Процесс повторяется, пока не будет найден элемент с заданным ключом или пока не будет определено, что такого элемента нет в таблице.

2. Вставка:

- Для вставки нового элемента в отсортированную таблицу сначала определяется место для вставки с помощью бинарного поиска.
- Затем все элементы, начиная с найденной позиции и до конца таблицы, сдвигаются на одну позицию вправо, чтобы освободить место для нового элемента.
- Новый элемент вставляется на найденное место.

3. Удаление:

- Для удаления элемента из отсортированной таблицы сначала элемент находится с помощью бинарного поиска.
- После нахождения элемента его удаляют, а все последующие элементы сдвигаются на одну позицию влево, чтобы заполнить пустую позицию.
- Таким образом, отсортированность таблицы сохраняется после удаления элемента.

3.1.3 Хэш-таблица открытого перемешивания

Алгоритм хэш-таблицы с открытым перемешиванием (или просто открытой адресацией) предполагает разрешение коллизий путем поиска свободной ячейки в таблице для вставки элемента.

1. Хэширование:

- Для вставки элемента сначала вычисляется хэш-значение ключа элемента с помощью хэш-функции.
- Хэш-функция преобразует ключ в индекс таблицы, в котором элемент должен быть размещен.

2. Разрешение коллизий:

- Если элемент пытается вставиться в ячейку, которая уже занята другим элементом (коллизия), применяется метод открытого перемешивания для нахождения свободной ячейки.
- Один из наиболее распространенных методов открытого перемешивания - это метод линейного пробирования, при котором ищется следующая свободная ячейка путем последовательного пробирования по индексам таблицы.
- Другие методы включают квадратичное пробирование и двойное хэширование, каждый из которых имеет свои преимущества и недостатки.

3. Вставка:

- После вычисления хэш-значения элемента и разрешения возможной коллизии, элемент вставляется в найденную свободную ячейку.

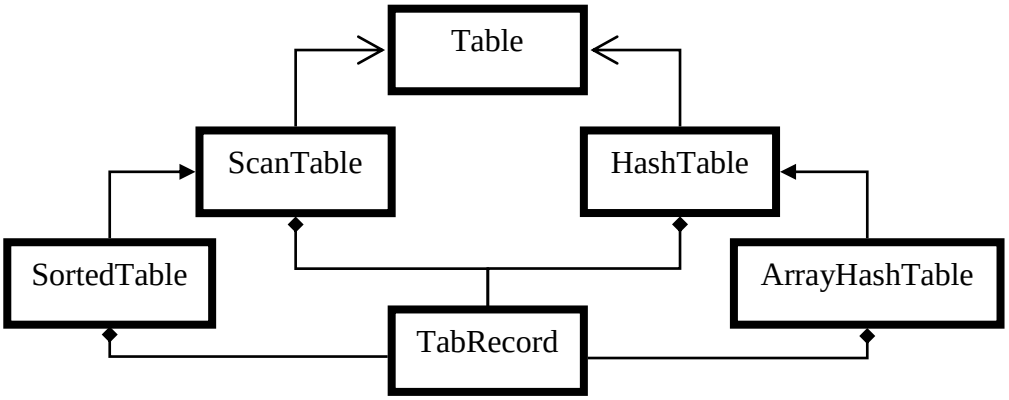
4. Поиск:

- Для поиска элемента сначала вычисляется хэш-значение ключа.
- Затем происходит последовательный поиск по ячейкам таблицы с использованием той же хэш-функции до тех пор, пока не будет найден элемент с соответствующим ключом или пока не будет обнаружена пустая ячейка, что означает отсутствие элемента в таблице.

5. Удаление:

- Для удаления элемента сначала он находится в таблице с помощью поиска по ключу.
- После нахождения элемента он удаляется из таблицы путем пометки соответствующей ячейки как свободной или удаления элемента, в зависимости от конкретной реализации.

3.2 Описание программной реализации



3.2.1 Описание класса TabRecord

```
template <class TKey, class TData>
struct TabRecord {

	TKey key;
	TData* data;

	TabRecord();
	TabRecord(const TKey& _key, TData* _data);
	TabRecord(const TabRecord<TKey, TData>& tr);
	~TabRecord();

	const TabRecord<TKey, TData>& operator=(const TabRecord<TKey, TData>&
tr);
};
```

Назначение: представление записи таблицы.

Поля:

key – ключ записи.

data – указатель на данные таблицы.

Конструкторы:

TabRecord();

Назначение: конструктор по умолчанию.

TabRecord(const TKey& _key, TData* _data);

Назначение: конструктор с параметром.

Входные параметры: **_key** – ключ, **_data** – указатель на данные.

TabRecord(const TabRecord<TKey, TData>& tr);

Назначение: конструктор копирования.

Входные параметры: `tr` – копируемый объект.

`~TabRecord();`

Назначение: деструктор.

Методы:

`const TabRecord<TKey, TData>& operator=(const TabRecord<TKey, TData>& tr);`

Назначение: оператор присваивания.

Входные параметры: `tr` – присваиваемы объект.

Выходные параметры: ссылка на себя.

3.2.2 Описание класса Table

```
template <class TKey, class TData>
class Table {
protected:
	int count;
	int max_size;
	int curr_pos;

public:
	Table(int _max_size = DEFAULT_SIZE);

	virtual void insert(const TKey& _key, TData* _data) = 0;
	virtual void remove(const TKey& _key) = 0;
	virtual TabRecord<TKey, TData>* find(const TKey& _key) = 0;
	virtual TabRecord<TKey, TData>* operator[](const TKey& _key) = 0;

	bool full() const noexcept;
	bool empty() const noexcept;
	bool ended() const noexcept;

	virtual bool reset() noexcept;
	virtual bool next() noexcept;

	int get_size() const noexcept;
	int get_max_size() const noexcept;
};
```

Назначение: абстрактное представление таблицы.

Поля:

`count` – количество записей в таблице.

`max_size` – максимальное число записей в таблице.

`curr_pos` – индекс текущей записи в таблице.

Конструкторы:

```
Table(int _max_size = DEFAULT_SIZE);
```

Назначение: конструктор с параметром.

Входные параметры: **_max_size** – максимальный размер таблицы.

Методы:

```
virtual void insert(const TKey& _key, TData* _data) = 0;
```

Назначение: вставка записи в таблицу.

Входные параметры: **_key** – ключ, **_data** – указатель на данные.

```
virtual void remove(const TKey& _key) = 0;
```

Назначение: удаление записи из таблицы по ключу.

Входные параметры: **_key** – ключ.

```
virtual TabRecord<TKey, TData>* find(const TKey& _key) = 0;
```

Назначение: поиск записи в таблице по ключу.

Входные параметры: **_key** – ключ.

Выходные параметры: указатель на запись.

```
virtual TabRecord<TKey, TData>* operator[](const TKey& _key) = 0;
```

Назначение: поиск записи в таблице по ключу.

Входные параметры: **_key** – ключ.

Выходные параметры: указатель на запись.

```
bool full() const noexcept;
```

Назначение: проверка, заполнена ли таблица.

Выходные параметры: **true** или **false**.

```
bool empty() const noexcept;
```

Назначение: проверка, пуста ли таблица.

Выходные параметры: **true** или **false**.

```
bool ended() const noexcept;
```

Назначение: проверка, является ли текущая запись в таблице последней.

Выходные параметры: **true** или **false**.

```
virtual bool reset() noexcept;
```

Назначение: переход в начало таблицы.

Выходные параметры: **true** или **false**.

```
virtual bool next() noexcept;
```

Назначение: переход на следующую запись таблицы.

Выходные параметры: **true** или **false**.

```
int get_size() const noexcept;
```

Назначение: получение числа записей в таблице.

Выходные параметры: количество записей в таблице.

```
int get_max_size() const noexcept;
```

Назначение: получение максимального числа записей в таблице.

Выходные параметры: максимальное количество записей в таблице.

3.2.3 Описание класса ScanTable

```
template <class TKey, class TData>
class ScanTable : public Table<TKey, TData> {
protected:
	TabRecord<TKey, TData>** recs;

public:
	ScanTable(int _max_size = DEFAULT_SIZE);
	ScanTable(const ScanTable<TKey, TData>& st);
	virtual ~ScanTable();

	virtual void insert(const TKey& _key, TData* _data);
	virtual void remove(const TKey& _key);

	virtual TabRecord<TKey, TData>* find(const TKey& key);
	TabRecord<TKey, TData>* operator[](const TKey& _key);

	friend std::ostream& operator<<(std::ostream& out, const ScanTable&
table);
};
```

Назначение: представление таблицы поиска.

Поля:

recs – память под указатели на записи.

Конструкторы:

```
ScanTable(int _max_size = DEFAULT_SIZE);
```

Назначение: конструктор с параметром.

Входные параметры: **_max_size** – максимальный размер таблицы.

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

Назначение: конструктор копирования.

Входные параметры: **st** – копируемый объект.

virtual ~ScanTable();

Назначение: деструктор.

Методы:

irtual void insert(const TKey& _key, TData* _data);

Назначение: вставка записи в таблицу.

Входные параметры: **_key** – ключ, **_data** – указатель на данные.

virtual void remove(const TKey& _key);

Назначение: удаление записи из таблицы по ключу.

Входные параметры: **_key** – ключ.

virtual TabRecord<TKey, TData>* find(const TKey& key);

Назначение: поиск записи в таблице по ключу.

Входные параметры: **_key** – ключ.

Выходные параметры: указатель на запись.

TabRecord<TKey, TData>* operator[](const TKey& _key);

Назначение: поиск записи в таблице по ключу.

Входные параметры: **_key** – ключ.

Выходные параметры: указатель на запись.

friend std::ostream& operator<<(std::ostream& out, const ScanTable& table);

Назначение: печать таблицы.

Входные параметры: out – поток вывода, table – печатаемая таблица.

Выходные параметры: поток вывода.

3.2.4 Описание класса SortedTable

```
template <class TKey, class TData>
class SortedTable : public ScanTable<TKey, TData> {
private:
	void sort();
	int partition(int low, int high);
	void quick_sort(int low, int high);

public:
	SortedTable(int _max_size = DEFAULT_SIZE);
	SortedTable(const ScanTable<TKey, TData>& st);
	SortedTable(const SortedTable<TKey, TData>& srt);

	TabRecord<TKey, TData>* find(const TKey& key);
	TabRecord<TKey, TData>* operator[](const TKey& _key);
	void insert(const TKey& _key, TData* _data);
	void remove(const TKey& _key);

	friend std::ostream& operator<<(std::ostream& out, const SortedTable&
table);
};
```

Назначение: представление отсортированной таблицы

Конструкторы:

`SortedTable(int _max_size = DEFAULT_SIZE);`

Назначение: конструктор с параметром.

Входные параметры: **`_max_size`** – максимальный размер таблицы.

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

Назначение: конструктор преобразования типов.

Входные параметры: **`st`** – копируемый объект.

`SortedTable(const SortedTable<TKey, TData>& srt);`

Назначение: конструктор копирования.

Входные параметры: **`st`** – копируемый объект.

Методы:

`TabRecord<TKey, TData>* find(const TKey& key);`

Назначение: поиск записи в таблице по ключу.

Входные параметры: **`_key`** – ключ.

Выходные параметры: указатель на запись.

`TabRecord<TKey, TData>* operator[](const TKey& _key);`

Назначение: поиск записи в таблице по ключу.

Входные параметры: **`_key`** – ключ.

Выходные параметры: указатель на запись.

```
void insert(const TKey& _key, TData* _data);
```

Назначение: вставка записи в таблицу.

Входные параметры: **_key** – ключ, **_data** – указатель на данные.

```
void remove(const TKey& _key);
```

Назначение: удаление записи из таблицы по ключу.

Входные параметры: **_key** – ключ.

```
friend std::ostream& operator<<(std::ostream& out, const SortedTable& table)
```

Назначение: печать таблицы.

Входные параметры: out – поток вывода, table – печатаемая таблица.

Выходные параметры: поток вывода.

```
void sort();
```

Назначение: сортировка.

```
void quick_sort(int low, int high);
```

Назначение: быстрая сортировка.

Входные параметры: **low** – левый индекс, **high** – правый индекс.

```
int partition(int low, int high);
```

Назначение: разбиение массива.

Входные параметры: **low** – левый индекс, **high** – правый индекс.

Выходные параметры: индекс опорного элемента.

3.2.5 Описание класса HashTable

```
template <class TKey, class TData>
class HashTable : public Table<TKey, TData> {
protected:
	virtual size_t hash_func(const TKey& key) = 0;

public:
	HashTable(size_t size = DEFAULT_SIZE) : Table(size) {};
};
```

Назначение: абстрактное представление хэш-таблицы.

Конструкторы:

`HashTable(size_t size = DEFAULT_SIZE);`

Назначение: конструктор с параметром.

Входные параметры: **`_max_size`** – максимальный размер таблицы.

Методы:

`virtual size_t hash_func(const TKey& key) = 0;`

Назначение: хэш-функция.

Входные параметры: **`_key`** – ключ.

Выходные параметры: хэш.

3.2.6 Описание класса ArrayHashTable

```
template <class TKey, class TData>
class ArrayHashTable : public HashTable<TKey, TData> {
protected:
	size_t hash_step;
	TabRecord<TKey, TData>** recs;
	TabRecord<TKey, TData>* pMark;
	int free_pos_ind;

	size_t hash_func(const TKey& key);
	size_t get_next_pos(size_t ind);

	void coprime_check(size_t _max_size, size_t _hash_step);
public:
	ArrayHashTable(size_t _max_size = DEFAULT_SIZE, size_t _hash_step =
DEFAULT_HASHSTEP);
	ArrayHashTable(const ArrayHashTable<TKey, TData>& aht);
	virtual ~ArrayHashTable();

	TabRecord<TKey, TData>* find(const TKey& key);
	TabRecord<TKey, TData>* operator[](const TKey& _key);
	void insert(const TKey& _key, TData* _data);
	void remove(const TKey& _key);

	bool reset() noexcept;
	bool next() noexcept;

	friend std::ostream& operator<<(std::ostream& out, const ArrayHashTable&
table);
};
```

Назначение: представление хэш-таблицы открытого перемешивания.

Поля:

`hash_step` – шаг хэша.

`recs` – память под указатели на записи.

`pMark` – запись-метка.

`free_pos_ind` – индекс свободной позиции.

Конструкторы:

```
ArrayHashTable(size_t _max_size = DEFAULT_SIZE, size_t _hash_step = DEFAULT_HASHSTEP);
```

Назначение: конструктор с параметром.

Входные параметры: **_max_size** – максимальный размер таблицы, **_hash_step** – шаг хэша.

```
ArrayHashTable(const ArrayHashTable<TKey, TData>& aht);
```

Назначение: конструктор копирования.

Входные параметры: **aht** – копируемый объект.

```
virtual ~ArrayHashTable();
```

Назначение: деструктор.

Методы:

```
TabRecord<TKey, TData>* find(const TKey& key);
```

Назначение: поиск записи в таблице по ключу.

Входные параметры: **_key** – ключ.

Выходные параметры: указатель на запись.

```
TabRecord<TKey, TData>* operator[](const TKey& _key);
```

Назначение: поиск записи в таблице по ключу.

Входные параметры: **_key** – ключ.

Выходные параметры: указатель на запись.

```
void insert(const TKey& _key, TData* _data);
```

Назначение: вставка записи в таблицу.

Входные параметры: **_key** – ключ, **_data** – указатель на данные.

```
void remove(const TKey& _key);
```

Назначение: удаление записи из таблицы по ключу.

Входные параметры: **_key** – ключ.

```
bool reset() noexcept;
```

Назначение: переход в начало таблицы.

Выходные параметры: **true** или **false**.

```
bool next() noexcept;
```

Назначение: переход на следующую запись таблицы.

Выходные параметры: **true** или **false**.

```
size_t hash_func(const TKey& key);
```

Назначение: хэш-функция.

Входные параметры: **_key** – ключ.

Выходные параметры: хэш.

```
size_t get_next_pos(size_t ind);
```

Назначение: получение следующего индекса.

Входные параметры: **ind** – индекс.

Выходные параметры: следующий индекс.

```
friend std::ostream& operator<<(std::ostream& out, const ArrayHashTable& table);
```

Назначение: печать таблицы.

Входные параметры: out – поток вывода, table – печатаемая таблица.

Выходные параметры: поток вывода.

```
void coprime_check(size_t _max_size, size_t _hash_step);
```

Назначение: проверка на взаимную простоту чисел.

Входные параметры: **_max_size** – максимальный размер таблицы, **_hash_step** – шаг хэша.

Заключение

В заключении данной лабораторной работы хочется подчеркнуть важность разработки и реализации эффективных структур данных для хранения и обработки информации. В ходе выполнения этой работы были изучены и реализованы различные классы таблиц, включая таблицы с линейным поиском, сортированные таблицы с бинарным поиском и хэш-таблицы с открытым перемешиванием.

Благодаря разнообразию структур данных и методов их реализации, была освоена широкая область алгоритмов и структур данных, что позволяет эффективно решать различные задачи хранения и обработки данных. В частности, использование этих структур для хранения и обработки полиномов позволяет значительно упростить и оптимизировать операции над ними.

В результате успешного выполнения этой лабораторной работы были получены полезные навыки разработки и тестирования структур данных, которые могут быть применены в различных областях программирования, где требуется эффективная работа с данными.

Литература

1. Лекция «Списковые структуры хранения» Сысоева А.В. https://cloud.unn.ru/s/x33MEa9on8HgNgw
2. Лекция «Списки в динамической памяти» Сысоева А.В. https://cloud.unn.ru/s/rCiKGSX33SSGPi4
3. Лекция «Полиномы» Сысоева А.В. https://cloud.unn.ru/s/t6o9kp5g9bpf2yz
4. Лекция «Организация доступа по имени. Таблицы» Сысоева А.В. https://cloud.unn.ru/s/2Y92XyGc7r3XdBC
5. Лекция «Хеш-таблицы» Сысоева А.В. https://cloud.unn.ru/s/B5fr3gKAL2LoHyH

Приложения

Приложение А. Реализация класса TabRecord

```
template <class TKey, class TData>
struct TabRecord {

	TKey key;
	TData* data;

	TabRecord();
	TabRecord(const TKey& _key, TData* _data);
	TabRecord(const TabRecord<TKey, TData>& tr);
	~TabRecord();

	const TabRecord<TKey, TData>& operator=(const TabRecord<TKey, TData>&
tr);
};

template <class TKey, class TData>
TabRecord<TKey, TData>::TabRecord() : key(), data(nullptr) {};

template <class TKey, class TData>
TabRecord<TKey, TData>::TabRecord(const TKey& _key, TData* _data) {
	key = _key;
	data = new TData(*_data);
}

template <class TKey, class TData>
TabRecord<TKey, TData>::TabRecord(const TabRecord<TKey, TData>& tr) {
	key = tr.key;
	data = new TData(*tr.data);
}

template <class TKey, class TData>
TabRecord<TKey, TData>::~TabRecord() {
	if (data) delete data;
}

template <class TKey, class TData>
const TabRecord<TKey, TData>& TabRecord<TKey, TData>::operator=(const
TabRecord<TKey, TData>& tr) {
	key = tr.key;

	if (data) delete data;
	data = new TData(*tr.data);

	return (*this);
}
```

Приложение Б. Реализация класса Table

```
#define DEFAULT_SIZE 101

template <class TKey, class TData>
class Table {
protected:
	int count;
	int max_size;
	int curr_pos;

public:
```

```
	Table(int _max_size = DEFAULT_SIZE);

	virtual void insert(const TKey& _key, TData* _data) = 0;
	virtual void remove(const TKey& _key) = 0;
	virtual TabRecord<TKey, TData>* find(const TKey& _key) = 0;
	virtual TabRecord<TKey, TData>* operator[](const TKey& _key) = 0;

	bool full() const noexcept;
	bool empty() const noexcept;
	bool ended() const noexcept;

	virtual bool reset() noexcept;
	virtual bool next() noexcept;

	int get_size() const noexcept;
	int get_max_size() const noexcept;
};

template <class TKey, class TData>
Table<TKey, TData>::Table(int _max_size) {
	max_size = _max_size;
	count = 0;
	curr_pos = -1;
}

template <class TKey, class TData>
bool Table<TKey, TData>::full() const noexcept {
	return (count == max_size);
}

template <class TKey, class TData>
bool Table<TKey, TData>::empty() const  noexcept {
	return (count == 0);
}

template <class TKey, class TData>
bool Table<TKey, TData>::ended() const noexcept {
	return (curr_pos >= max_size);
}


template <class TKey, class TData>
bool Table<TKey, TData>::reset() noexcept {
	if (!empty()) {
		curr_pos = 0;
		return ended();
	}
	else {
		curr_pos = -1;
		return ended();
	}
}

template <class TKey, class TData>
bool Table<TKey, TData>::next() noexcept {
	curr_pos++;
	return ended();
}


template <class TKey, class TData>
int Table<TKey, TData>::get_size() const noexcept {
	return count;
```

```
}

template <class TKey, class TData>
int Table<TKey, TData>::get_max_size() const noexcept {
	return max_size;
}
```

Приложение В. Реализация класса ScanTable

```
template <class TKey, class TData>
class ScanTable : public Table<TKey, TData> {
protected:
	TabRecord<TKey, TData>** recs;

public:
	ScanTable(int _max_size = DEFAULT_SIZE);
	ScanTable(const ScanTable<TKey, TData>& st);
	virtual ~ScanTable();

	virtual void insert(const TKey& _key, TData* _data);
	virtual void remove(const TKey& _key);

	virtual TabRecord<TKey, TData>* find(const TKey& key);
	TabRecord<TKey, TData>* operator[](const TKey& _key);

	friend std::ostream& operator<<(std::ostream& out, const ScanTable&
table) {
		out << "Table size: " << table.count << endl;
		for (int i = 0; i < table.count; i++) {
			if (table.recs[i]) {
				out << "(" << table.recs[i]->key << ", " <<
*table.recs[i]->data << "); " << endl;
			}
		}
		return out;
	}
};


template <class TKey, class TData>
ScanTable<TKey, TData>::ScanTable(int _max_size) {
	if (_max_size <= 0) {
		std::string exp = "ERROR: Table max_size cant be less or equal
than 0.";
		throw exp;
	}

	max_size = _max_size;
	recs = new TabRecord<TKey, TData>* [max_size];

	count = 0;
	curr_pos = -1;

}

template <class TKey, class TData>
ScanTable<TKey, TData>::ScanTable(const ScanTable<TKey, TData>& st) {
	max_size = st.max_size;
	count = st.count;
	curr_pos = st.curr_pos;

	recs = new TabRecord<TKey, TData>* [max_size];
	for (int i = 0; i < count; i++) {
```

```
            TKey key = st.recs[i]->key;
            TData* data = st.recs[i]->data;
            recs[i] = new TabRecord<TKey, TData>(key, data);
      }
}

template <class TKey, class TData>
ScanTable<TKey, TData>::~ScanTable() {
      if (recs != nullptr) {
            for (int i = 0; i < count; i++)
                  if (recs[i] != nullptr) delete recs[i];
            delete recs;
      }

}


template <class TKey, class TData>
void ScanTable<TKey, TData>::insert(const TKey& _key, TData* _data) {
      if (full()) {
            std::string exp = "ERROR: Table is full.";
            throw exp;
      }

      if (find(_key) == nullptr) {
            recs[count] = new TabRecord<TKey, TData>(_key, _data);
            count++;
      }
}

template <class TKey, class TData>
void ScanTable<TKey, TData>::remove(const TKey& _key) {
      if (empty()) {
            std::string exp = "ERROR: Table is empty.";
            throw exp;
      }

      if (find(_key) != nullptr) {
            delete recs[curr_pos];
            for (int i = curr_pos; i < count; i++) {
                  recs[i] = recs[i + 1];
            }
            count--;
      }
      else {
            std::string exp = "ERROR: key not found.";
            throw exp;
      }
}

template <class TKey, class TData>
TabRecord<TKey, TData>* ScanTable<TKey, TData>::find(const TKey& key) {
      TabRecord<TKey, TData>* res = nullptr;

      for (int i = 0; i < count; i++) {
            if (recs[i]->key == key) {
                  curr_pos = i;
                  res = recs[i];
                  break;
            }
      }

      return res;
```

```
}

template <class TKey, class TData>
TabRecord<TKey, TData>* ScanTable<TKey, TData>::operator[](const TKey& _key)
{
	return find(_key);
}
```

Приложение Г. Реализация класса SortedTable

```
template <class TKey, class TData>
class SortedTable : public ScanTable<TKey, TData> {
private:
	void sort();
	int partition(int low, int high);
	void quick_sort(int low, int high);

public:
	SortedTable(int _max_size = DEFAULT_SIZE);
	SortedTable(const ScanTable<TKey, TData>& st);
	SortedTable(const SortedTable<TKey, TData>& srt);

	TabRecord<TKey, TData>* find(const TKey& key);
	TabRecord<TKey, TData>* operator[](const TKey& _key);
	void insert(const TKey& _key, TData* _data);
	void remove(const TKey& _key);

	friend std::ostream& operator<<(std::ostream& out, const SortedTable&
table) {
		out << "Table size: " << table.count << endl;
		for (int i = 0; i < table.count; i++) {
			if (table.recs[i]) {
				out << "(" << table.recs[i]->key << ", " <<
*table.recs[i]->data << "); " << endl;
			}
		}
		return out;
	}
};


template <class TKey, class TData>
void SortedTable<TKey, TData>::sort() {
	quick_sort(0, count - 1);
}

template <class TKey, class TData>
int SortedTable<TKey, TData>::partition(int low, int high) {
	TKey pivot = recs[high]->key;
	int i = low - 1;

	for (int j = low; j <= high - 1; ++j) {
		if (recs[j]->key < pivot) {
			++i;
			std::swap(recs[i], recs[j]);
		}
	}
	std::swap(recs[i + 1], recs[high]);
	return (i + 1);
}

template <class TKey, class TData>
void SortedTable<TKey, TData>::quick_sort(int low, int high) {
```

```
	if (low < high) {
		int pi = partition(low, high);

		quick_sort(low, pi - 1);
		quick_sort(pi + 1, high);
	}
}


template <class TKey, class TData>
SortedTable<TKey, TData>::SortedTable(int _max_size) : ScanTable(_max_size)
{}

template <class TKey, class TData>
SortedTable<TKey, TData>::SortedTable(const ScanTable<TKey, TData>& st) :
ScanTable(st) {
	sort();
}

template <class TKey, class TData>
SortedTable<TKey, TData>::SortedTable(const SortedTable<TKey, TData>& srt) {
	count = srt.count;
	max_size = srt.max_size;
	curr_pos = srt.max_size;

	recs = new TabRecord<TKey, TData>* [max_size];
	for (int i = 0; i < count; i++) {
		recs[i] = new TabRecord<TKey, TData>(*srt.recs[i]);
	}
}


template <class TKey, class TData>
TabRecord<TKey, TData>* SortedTable<TKey, TData>::find(const TKey& key) {
	int left = 0, right = count - 1;
	TabRecord<TKey, TData>* search = nullptr;

	while (left <= right) {
		int mid = (right + left) / 2;

		if (recs[mid]->key == key) {
			search = recs[mid];
			right = mid;
			left = mid + 1;
		}
		else if (recs[mid]->key < key) left = mid + 1;
		else right = mid - 1;
	}
	curr_pos = right;
	return search;
}

template <class TKey, class TData>
TabRecord<TKey, TData>* SortedTable<TKey, TData>::operator[](const TKey&
_key) {
	return find(_key);
}

template <class TKey, class TData>
void SortedTable<TKey, TData>::insert(const TKey& _key, TData* _data) {
	if (full()) {
		std::string exp = "ERROR: Table is full.";
		throw exp;
```

```
	}

	if (find(_key) == nullptr) {
		for (int i = count - 1; i >= curr_pos; i--) {
			recs[i + 1] = recs[i];
		}
		recs[curr_pos + 1] = new TabRecord<TKey, TData>(_key, _data);
		count++;
	}
}

template <class TKey, class TData>
void SortedTable<TKey, TData>::remove(const TKey& _key) {
	if (empty()) {
		std::string exp = "ERROR: Table is empty.";
		throw exp;
	}

	TabRecord<TKey, TData>* rec = find(_key);
	if (rec == nullptr) {
		std::string exp = "ERROR: Key nod found.";
		throw exp;
	}
	else {
		delete rec;
		for (int i = curr_pos; i < count - 1; i++) {
			recs[i] = recs[i + 1];
		}
		count--;
	}
}
```

Приложение Д. Реализация класса HashTable

```
template <class TKey, class TData>
class HashTable : public Table<TKey, TData> {
protected:
	virtual size_t hash_func(const TKey& key) = 0;

public:
	HashTable(size_t size = DEFAULT_SIZE) : Table(size) {};
};
```

Приложение Е. Реализация класса ArrayHashTable

```
#define DEFAULT_HASHSTEP 10

template <class TKey, class TData>
class ArrayHashTable : public HashTable<TKey, TData> {
protected:
	size_t hash_step;
	TabRecord<TKey, TData>** recs;
	TabRecord<TKey, TData>* pMark;
	int free_pos_ind;

	size_t hash_func(const TKey& key);
	size_t get_next_pos(size_t ind);

	void coprime_check(size_t _max_size, size_t _hash_step);
public:
	ArrayHashTable(size_t _max_size = DEFAULT_SIZE, size_t _hash_step =
DEFAULT_HASHSTEP);
	ArrayHashTable(const ArrayHashTable<TKey, TData>& aht);
```

```
	virtual ~ArrayHashTable();

	TabRecord<TKey, TData>* find(const TKey& key);
	TabRecord<TKey, TData>* operator[](const TKey& _key);
	void insert(const TKey& _key, TData* _data);
	void remove(const TKey& _key);

	bool reset() noexcept;
	bool next() noexcept;

	friend std::ostream& operator<<(std::ostream& out, const ArrayHashTable&
table) {
		out << "Table size: " << table.count << endl;
		for (int i = 0; i < table.max_size; i++) {
			if (table.recs[i] && table.recs[i] != table.pMark) {
				out << "(" << table.recs[i]->key << ", " <<
*table.recs[i]->data << "); " << endl;
			}
		}
		return out;
	}
};


template <class TKey, class TData>
size_t ArrayHashTable<TKey, TData>::hash_func(const TKey& key) {
	std::hash<TKey> hasher;
	return hasher(key) % max_size;
}

template <class TKey, class TData>
size_t ArrayHashTable<TKey, TData>::get_next_pos(size_t ind) {
	return (ind + hash_step) % max_size;
}

template <class TKey, class TData>
void ArrayHashTable<TKey, TData>::coprime_check(size_t _max_size, size_t
_hash_step) {
	if (_max_size <= 0) {
		std::string exp = "ERROR: Table max_size cant be less or equal
than 0.";
		throw exp;
	}

	if (_hash_step <= 0) {
		std::string exp = "ERROR: Table hash_step cant be less or equal
than 0.";
		throw exp;
	}

	// gcd check
	size_t a = _max_size;
	size_t b = _hash_step;
	while (b != 0) {
		size_t tmp = b;
		b = a % b;
		a = tmp;
	}
	if (a != 1) {
		throw std::string("ERROR: max_size and hash_step must be coprime
numbers.");
	}
}
```

```
template <class TKey, class TData>
ArrayHashTable<TKey,    TData>::ArrayHashTable(size_t    _max_size,    size_t
_hash_step) : HashTable(_max_size) {
      coprime_check(_max_size, _hash_step);

      recs = new TabRecord<TKey, TData>* [max_size];
      hash_step = _hash_step;
      pMark = new TabRecord<TKey, TData>();
      free_pos_ind = -1;

      for (int i = 0; i < max_size; i++) recs[i] = nullptr;
}

template <class TKey, class TData>
ArrayHashTable<TKey,    TData>::ArrayHashTable(const    ArrayHashTable<TKey,
TData>& aht) {
      max_size = aht.max_size;
      count = aht.count;
      hash_step = aht.hash_step;
      curr_pos = aht.curr_pos;
      free_pos_ind = aht.free_pos_ind;

      recs = new TabRecord<TKey, TData>* [max_size];
      pMark = new TabRecord<TKey, TData> ();

      for (int i = 0; i < max_size; i++) {
            TabRecord<TKey, TData>* tmp = aht.recs[i];

            if (tmp == nullptr)     recs[i] = tmp;
            else if (tmp == aht.pMark)    recs[i] = pMark;
            else recs[i] = new TabRecord<TKey, TData>(*tmp);
      }
}

template <class TKey, class TData>
ArrayHashTable<TKey, TData>::~ArrayHashTable() {
      for (int i = 0; i < max_size; i++) {
            if (recs[i] != nullptr && recs[i] != pMark) delete recs[i];
      }
      if (recs) delete[] recs;
      if (pMark) delete pMark;
}


template <class TKey, class TData>
TabRecord<TKey, TData>* ArrayHashTable<TKey, TData>::find(const TKey& key) {
      curr_pos = hash_func(key);
      for (int i = 0; i < max_size; i++) {
            if (recs[curr_pos] == nullptr)
                  return nullptr;

            else if (recs[curr_pos] == pMark && free_pos_ind == -1)
                  free_pos_ind = curr_pos;

            else if (recs[curr_pos]->key == key)
                  return recs[curr_pos];

            curr_pos = get_next_pos(curr_pos);
      }

      return nullptr;
```

```
}

template <class TKey, class TData>
TabRecord<TKey, TData>* ArrayHashTable<TKey, TData>::operator[](const TKey&
_key) {
	return find(_key);
}

template <class TKey, class TData>
void ArrayHashTable<TKey, TData>::insert(const TKey& _key, TData* _data) {
	if (full()) {
		std::string exp = "ERROR: table is full.";
		throw exp;
	}

	TabRecord<TKey, TData>* rec = find(_key);
	if (rec != nullptr && rec->key == _key) return;

	if (rec != nullptr && free_pos_ind != -1) {
		curr_pos = free_pos_ind;
	}
	recs[curr_pos] = new TabRecord<TKey, TData>(_key, _data);
	count++;
}

template <class TKey, class TData>
void ArrayHashTable<TKey, TData>::remove(const TKey& _key) {
	TabRecord<TKey, TData>* tmp = find(_key);

	if (tmp == nullptr) {
		std::string exp = "ERROR: key not found";
		throw exp;
	}

	delete tmp;
	recs[curr_pos] = pMark;
	free_pos_ind = -1;
	count--;
}


template <class TKey, class TData>
bool ArrayHashTable<TKey, TData>::reset() noexcept {
	if (!empty()) {
		curr_pos = 0;
		return ended();
	}
	else {
		curr_pos = -1;
		return ended();
	}
}

template <class TKey, class TData>
bool ArrayHashTable<TKey, TData>::next() noexcept {
	if (curr_pos < max_size && !empty()) {
		curr_pos++;
		return ended();
	}
	else return ended();
}
```

Приложение Ж. Программа демонстрации таблиц полиномов

```
#include <iostream>
#include <string>
#include "tableslib.h"
#include "tpolynom.h"

using namespace std;


#define EXIT   0
#define INSERT 1
#define DELETE 2
#define OPERATIONS 3
#define PRINT  4
#define SELECT_TABLES 5

#define SUM 1
#define SUB 2
#define MUL 3
#define DIFF_X 4
#define DIFF_Y 5
#define DIFF_Z 6

#define SCAN 1
#define SORTED 2
#define ARRAY_HASH 3

ScanTable     <string, TPolynom> scan_table            (DEFAULT_SIZE);
SortedTable   <string, TPolynom> sorted_table    (DEFAULT_SIZE);
ArrayHashTable<string,         TPolynom>         array_hash_table(DEFAULT_SIZE,
DEFAULT_HASHSTEP);

TabRecord<string, TPolynom>* tr_scan         = nullptr;
TabRecord<string, TPolynom>* tr_sorted       = nullptr;
TabRecord<string, TPolynom>* tr_array_hash = nullptr;

bool flag_scan          = true;
bool flag_sorted  = true;
bool flag_array_hash= true;


void menu();
void insert();
void remove();
void operations();
void print_tables();
void select_tables();
void print_selected_tables();


int main() {

      try {

            menu();

      }
      catch (string exp) {
            cout << exp << endl;
            return -1;
```

```
	}
	return 0;
}


void menu() {
	int ans;

	while (1) {
		system("cls");
		cout << "PATH:menu\\" << endl;
		print_selected_tables();
		cout << endl;

		cout << "1. Insert polynom" << endl;
		cout << "2. Remove polynom" << endl;
		cout << "3. Operations" << endl;
		cout << "4. Print tables" << endl;
		cout << "5. Select tables" << endl;
		cout << "0. EXIT" << endl << endl;


		cin >> ans;

		if (ans == INSERT) {
			insert();
		}

		else if (ans == DELETE) {
			remove();
		}

		else if (ans == OPERATIONS) {
			operations();
		}

		else if (ans == PRINT) {
			print_tables();
		}

		else if (ans == SELECT_TABLES) {
			select_tables();
		}

		else if (ans == EXIT) {
			return;
		}
	}
}

void insert() {
	system("cls");
	cout << "PATH:menu\\insertion\\" << endl;

	if (!(flag_scan || flag_sorted || flag_array_hash)) {
		cout << "Tables not selected." << endl;
		system("pause");
		return;
	}
	print_selected_tables();
	cout << endl;
```

```
	cout << "Enter the polynomial you want to add to the tables." << endl <<
endl;


	try {
		string name;
		cin.ignore(numeric_limits<streamsize>::max(), '\n');
		getline(cin, name);

		TPolynom new_polynom(name);
		string new_polynom_name = new_polynom.get_string();

		cout << endl;

		if (flag_scan) {
			try {
				scan_table.insert(new_polynom_name, &new_polynom);
				cout << "ScanTable.insert(): OK" << endl;
			}
			catch (string exp) {
				cout << "ScanTable.insert(): " << exp << endl;
			}
		}

		if (flag_sorted) {
			try {
				sorted_table.insert(new_polynom_name, &new_polynom);
				cout << "SortedTable.insert(): OK" << endl;
			}
			catch (string exp) {
				cout << "SortedTable.insert(): " << exp << endl;
			}
		}

		if (flag_array_hash) {
			try {
				array_hash_table.insert(new_polynom_name,
&new_polynom);
				cout << "ArrayHashTable.insert(): OK" << endl;
			}
			catch (string exp) {
				cout << "ArrayHashTable.insert(): " << exp << endl;
			}
		}
	}
	catch (string exp) {
		cout << exp << endl;

		system("pause");
	}

	system("pause");
}

void remove() {
	system("cls");
	cout << "PATH:menu\\removing\\" << endl;

	if (!(flag_scan || flag_sorted || flag_array_hash)) {
		cout << "Tables not selected." << endl;
		system("pause");
		return;
	}
```

```
	print_selected_tables();
	cout << endl;

	cout << "Enter the polynomial you want to remove from the table." << endl << endl;


	try {
		string name;
		cin.ignore(numeric_limits<streamsize>::max(), '\n');
		getline(cin, name);

		TPolynom polynom(name);
		string polynom_name = polynom.get_string();

		cout << endl;

		if (flag_scan) {
			try {
				scan_table.remove(polynom_name);
				cout << "ScanTable.remove(): OK" << endl;
			}
			catch (string exp) {
				cout << "ScanTable.remove(): " << exp << endl;
			}
		}

		if (flag_sorted) {
			try {
				sorted_table.remove(polynom_name);
				cout << "SortedTable.remove(): OK" << endl;
			}
			catch (string exp) {
				cout << "SortedTable.remove(): " << exp << endl;
			}
		}

		if (flag_array_hash) {
			try {
				array_hash_table.remove(polynom_name);
				cout << "ArrayHashTable.remove(): OK" << endl;
			}
			catch (string exp) {
				cout << "ArrayHashTable.remove(): " << exp << endl;
			}
		}
	}
	catch (string exp) {
		cout << exp << endl;

		system("pause");
		return;
	}

	system("pause");
}

void operations() {
	system("cls");
	cout << "PATH:menu\\operations\\" << endl;

	cout << "Select table." << endl << endl;
	cout << "0. none" << endl;
```

```
	cout << "1. ScanTable" << endl;
	cout << "2. SortedTable" << endl;
	cout << "3. ArrayHashTable" << endl << endl;

	int tab = 0;
	cin >> tab;
	if (tab < SCAN || tab > ARRAY_HASH) return;

	system("cls");
	cout << "PATH:menu\\operations\\" << endl;
	cout << "Selected table: ";
	if (tab == SCAN) cout << "ScanTable" << endl;
	if (tab == SORTED) cout << "SortedTable" << endl;
	if (tab == ARRAY_HASH) cout << "ArrayHashTable" << endl << endl;

	cout << "Select an operation to apply it on table polynomials." << endl
<< endl;
	cout << "0. none" << endl;
	cout << "1. +" << endl;
	cout << "2. -" << endl;
	cout << "3. *" << endl;
	cout << "4. diff_x" << endl;
	cout << "5. diff_y" << endl;
	cout << "6. diff_z" << endl << endl;

	int op = 0;
	cin >> op;
	if (op < SUM || op > DIFF_Z) return;

	system("cls");
	cout << "PATH:menu\\operations\\" << endl;
	TabRecord<string, TPolynom> *tab_polynom1 = nullptr, *tab_polynom2 =
nullptr;

	try {
		cout << "Enter the polynom to find it in the table: " << endl;

		string name1;
		cin.ignore(numeric_limits<streamsize>::max(), '\n');
		getline(cin, name1);
		TPolynom polynom1(name1);

		if (tab == SCAN) {
			tab_polynom1 = scan_table.find(polynom1.get_string());
		}
		if (tab == SORTED) {
			tab_polynom1 = sorted_table.find(polynom1.get_string());
		}
		if (tab == ARRAY_HASH) {
			tab_polynom1 = array_hash_table.find(polynom1.get_string());
		}
		if (tab_polynom1 == nullptr) {
			cout << "ERROR: polynom not found" << endl;
			system("pause");
			return;
		}
		cout << endl;

		if (op >= SUM && op <= MUL) {
			cout << "Enter the second polynom to find it in the table: "
<< endl;

			string name2;
```

```
                getline(cin, name2);
                TPolynom polynom2(name2);

                if (tab == SCAN) {
                        tab_polynom2 = scan_table.find(polynom2.get_string());
                }
                if (tab == SORTED) {
                        tab_polynom2                                                    =
sorted_table.find(polynom2.get_string());
                }
                if (tab == ARRAY_HASH) {
                        tab_polynom2                                                    =
array_hash_table.find(polynom2.get_string());
                }
                if (tab_polynom2 == nullptr) {
                        cout << "ERROR: second polynom not found" << endl;
                        system("pause");
                        return;
                }
        }
        cout << endl;

        system("cls");
        cout << "PATH:menu\\operations\\" << endl;
        TPolynom result("0");
        if (op == SUM) {
                result = *tab_polynom1->data + *tab_polynom2->data;
                cout << "polynom1 + polynom2 = " << result << endl << endl;
        }

        else if (op == SUB) {
                result = *tab_polynom1->data - *tab_polynom2->data;
                cout << "polynom1 - polynom2 = " << result << endl << endl;
        }

        else if (op == MUL) {
                result = *tab_polynom1->data * *tab_polynom2->data;
                cout << "polynom1 * polynom2 = " << result << endl << endl;
        }

        else if (op == DIFF_X) {
                result = tab_polynom1->data->diff_x();
                cout << "polynom.diff_x() = " << result << endl << endl;
        }

        else if (op == DIFF_Y) {
                result = tab_polynom1->data->diff_y();
                cout << "polynom.diff_x() = " << result << endl << endl;
        }

        else if (op == DIFF_Z) {
                result = tab_polynom1->data->diff_z();
                cout << "polynom.diff_x() = " << result << endl << endl;
        }

        cout << "Add result to this table? 1/0" << endl;
        int ans = 0;
        cin >> ans;
        if (ans) {
                try {
                        if (tab == SCAN) {
                                scan_table.insert(result.get_string(), &result);
                        }
```

```
                        if (tab == SORTED) {
                              sorted_table.insert(result.get_string(),
&result);
                        }
                        if (tab == ARRAY_HASH) {
                              array_hash_table.insert(result.get_string(),
&result);
                        }
                  }
                  catch (string exp) {
                        cout << exp << endl;
                  }
            }
            system("pause");
      }
      catch (string exp) {
            cout << exp << endl;
            system("pause");
            return;
      }
}

void print_tables() {
      system("cls");
      cout << "PATH:menu\\print\\" << endl;

      if (!(flag_scan || flag_sorted || flag_array_hash)) {
            cout << "Tables not selected." << endl;
            system("pause");
            return;
      }
      print_selected_tables();
      cout << endl;

      if (flag_scan) {
            cout << "ScanTable:" << endl;
            cout << scan_table << endl;
      }
      if (flag_sorted) {
            cout << "SortedTable:" << endl;
            cout << sorted_table << endl;
      }
      if (flag_array_hash) {
            cout << "ArrayHashTable:" << endl;
            cout << array_hash_table << endl;
      }

      system("pause");
}

void select_tables() {
      int st = 0;

      do {
            system("cls");
            cout << "# table_name     is_selected" << endl;
            cout << "1 ScanTable      " << flag_scan << endl;
            cout << "2 SortedTable    " << flag_sorted << endl;
            cout << "3 ArrayHashTable " << flag_array_hash << endl << endl;

            cout << "Select # of table to enable/disable." << endl;
            cout << "Press 0 to go back to previous menu." << endl << endl;
```

```
            cin >> st;

            if (st == 0) return;
            else if (st == 1) flag_scan = !flag_scan;
            else if (st == 2) flag_sorted = !flag_sorted;
            else if (st == 3) flag_array_hash = !flag_array_hash;

      } while (st != 0);
}

void print_selected_tables() {
      cout << "SELECTED TABLES: ";
      if (flag_scan) cout << "ScanTable ";
      if (flag_sorted) cout << "SortedTable ";
      if (flag_array_hash) cout << "ArrayHashTable ";
      cout << endl;
}
```