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

Отчёт по лабораторной работе
«Таблицы»

Выполнил(а): студент группы
3822Б1ФИ2
_____ / Иванов М. И./
Подпись

Проверил: преподаватель каф. МОСТ
ИИТММ
_____ / Арисова А. Н./
Подпись

Нижний Новгород
2024

Содержание

Введение

В лабораторной работе рассматривается вопрос разработки приложения, позволяющего хранить объекты типа полином в таблицах разных видов (упорядоченная, неупорядоченная и хеш-таблица), а также поверхностный анализ сложности выполнения операций вставки, поиска элемента по ключу и удаления элемента по ключу в каждом из представленных видов таблиц. В работе описывается устройство основных классов и методы этих классов, а также представлена демонстрация работы с пользовательским приложением.

1. Постановка задачи

Разработать программу, которая предоставляет возможность хранить данные в таблицах 3-х типов:

1)Неупорядоченная таблицы

2)Упорядоченная таблица

3)Хэш-таблица

В качестве данных использовать полиномы из лабораторной работы 4.

Особенности реализации:

1. Работа происходит сразу с таблицами всех типов.

2. Пользователь может положить в таблицу свой полином, с заданием ему некоторого наименования (ключа).

3. Пользователь может запросить (найти в/извлечь из) таблицы полином с заданным наименованием.

4. В ходе выполнения программы у пользователя должна оставаться возможность производить операции с полиномами, а так же размещать полученный результат в таблицу.

5. Во время работы программы происходит логирования количества произведенных операций. Пример: Пользователь ввел свой полином и запросил положить его в таблица, программа в ответ выводит на экран или в файл, какое действие произошло и сколько операций потребовалось на нее у разных типов таблиц. Аналогично с поиском и извлечением.

2. Руководство пользователя

При запуске приложения пользователю предлагается ввести свой первый моном, с которым дальше можно будет работать. На вход ожидается последовательный ввод вещественного коэффициента монома, показатель степени первой переменной „x“, второй переменной „y“ и третьей переменной „z“.

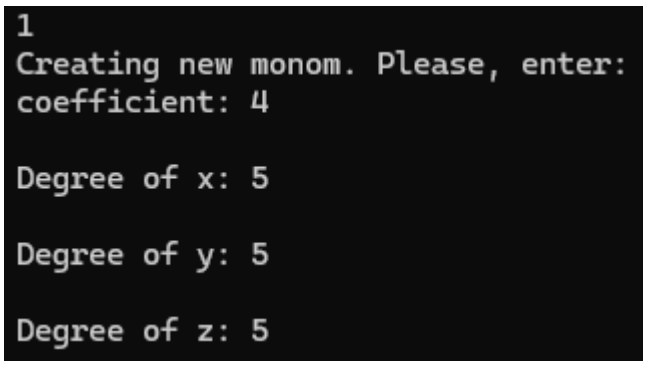
```
WELCOME to the POLINOM CREATOR !!!
Here you can add as much monoms and polinoms of 'xyz' view as you want and make some
 arithmetic operations with them (+, -, *).
Creating new monom. Please, enter:
coefficient: 2.5

Degree of x: 1

Degree of y: 2

Degree of z: 3
```

Далее пользователю предлагается сделать выбор из 6 опций.

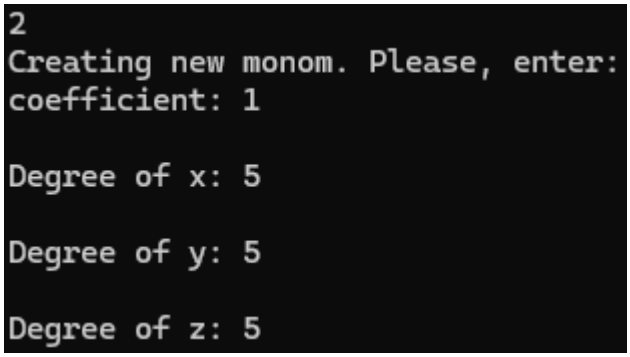
```
Choose an option beneath:
0 - FINISH THE PROGRAM
1 - CREATE NEW MONOM
2 - CREATE NEW POLINOM
3 - SAVE ACTIVE POLINOM
4 - SHOW EXISTING POLINOMS
5 - OPERATIONS WITH POLINOMS
```

Опция 0 - FINISH THE PROGRAM - Мгновенно завершает программу без сохранения данных.

Опция 1 – CREATE NEW MONOM – сохраняет активный моном в активный полином и запускает сценарий создания нового монома. На вход ожидается последовательный ввод вещественного коэффициента монома, показатель степени первой переменной „x“, второй переменной „y“ и третьей переменной „z“.

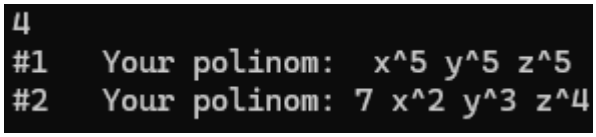


Опция 2 – CREATE NEW POLINOM – не сохраняет активный полином, создаёт новый пустой полином и запускает сценарий создания нового монома. На вход ожидается последовательный ввод вещественного коэффициента монома, показатель степени первой переменной „x“, второй переменной „y“ и третьей переменной „z“.

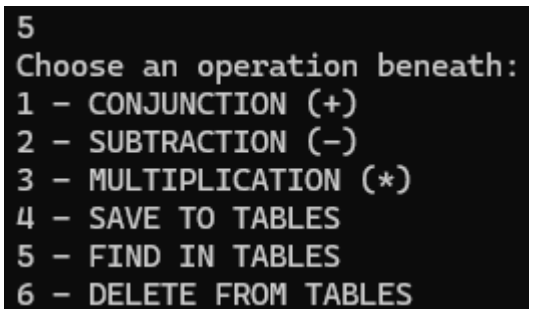


Опция 3 – SAVE ACTIVE POLINOM – сохраняет активный полином.

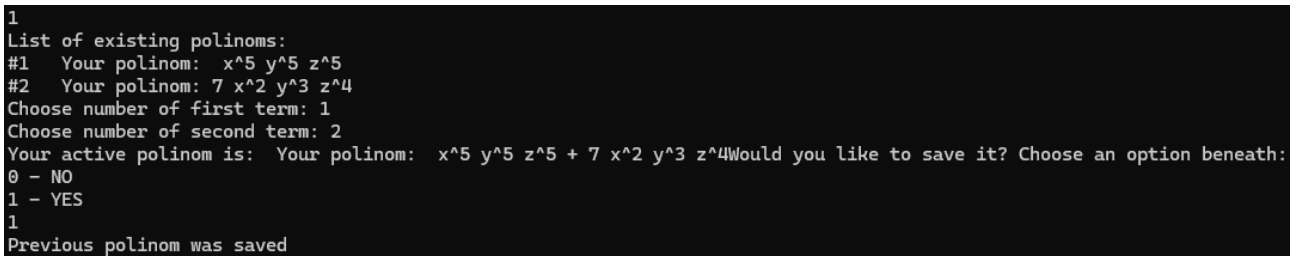
Опция 4 – SHOW EXISTING POLINOMS – выводит на консоль пронумерованный список из всех созданных полиномов.



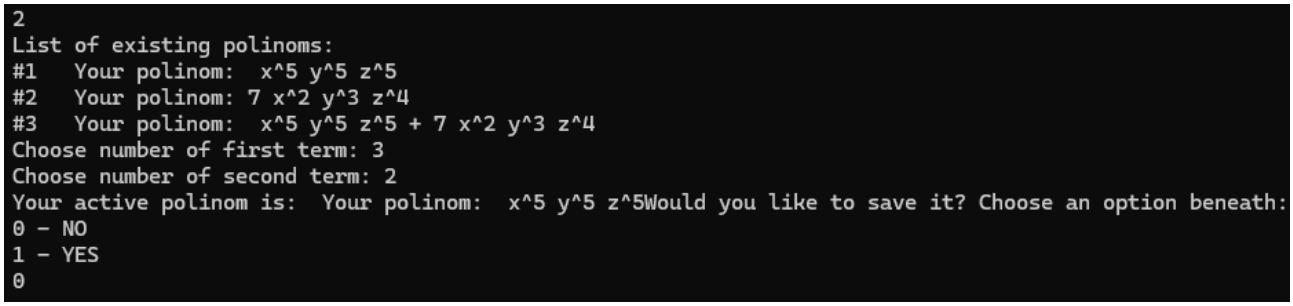
Опция 5 — OPERATIONS WITH POLINOMS – открывает меню доступных операций с полиномами.



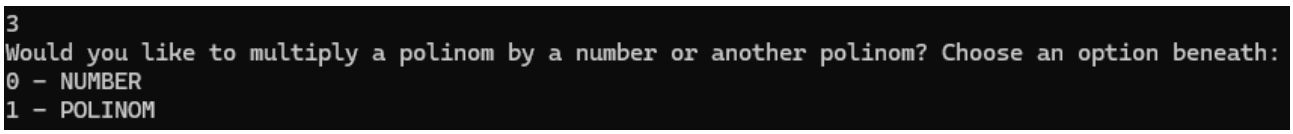
Опция 5.1 — CONJUNCTION (+) - запускает сценарий вывода на консоль всех полиномов. Ожидает на вход номер из списка первого слагаемого (полинома) и второго слагаемого. Выводит на экран результат сложения полиномов и предлагает сохранить его.



Опция 5.2 — SUBTRACTION (-) - запускает сценарий вывода на консоль всех полиномов. Ожидает на вход номер из списка уменьшаемого полинома и вычитаемого полинома. Выводит на экран результат вычитания полиномов и предлагает сохранить его.



Опция 5.3 — MULTIPLICATION (*) - открывает меню умножения.



Опция 5.3.1 — NUMBER - запускает сценарий вывода на консоль всех полиномов. Ожидает на вход номер из списка первого множителя (полинома) и вещественного числа. Выводит на экран результат умножения полинома на число и предлагает сохранить его.

```
0
List of existing polinoms:
#1   Your polinom:  x^5 y^5 z^5
#2   Your polinom: 7 x^2 y^3 z^4
#3   Your polinom:  x^5 y^5 z^5 + 7 x^2 y^3 z^4
Choose number of polinom: 1
Choose number to multiply: 10
Your active polinom is:  Your polinom: 10 x^5 y^5 z^5Would you like to save it? Choose an option beneath:
0 - NO
1 - YES
0
```

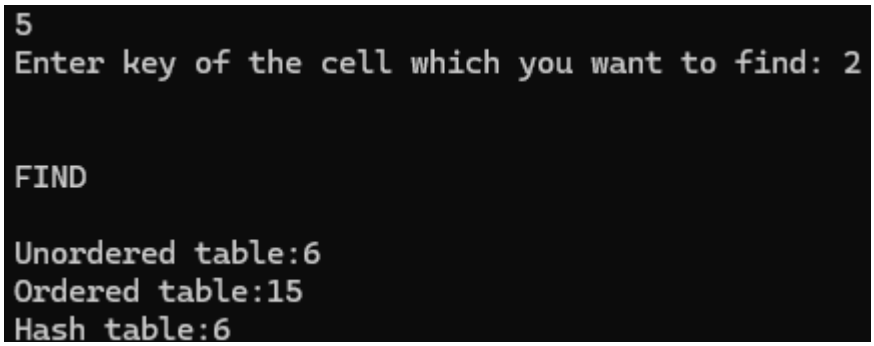
Опция 5.3.2 — POLINOM - запускает сценарий вывода на консоль всех полиномов. Ожидает на вход номер из списка первого множителя (полинома) и второго множителя (полинома). Выводит на экран результат умножения полиномов и предлагает сохранить его.

```
1
List of existing polinoms:
#1   Your polinom:  x^5 y^5 z^5
#2   Your polinom: 7 x^2 y^3 z^4
#3   Your polinom:  x^5 y^5 z^5 + 7 x^2 y^3 z^4
Choose number of first term: 1
Choose number of second term: 2
Your active polinom is:  Your polinom: 7 x^7 y^8 z^9Would you like to save it? Choose an option beneath:
0 - NO
1 - YES
0
```

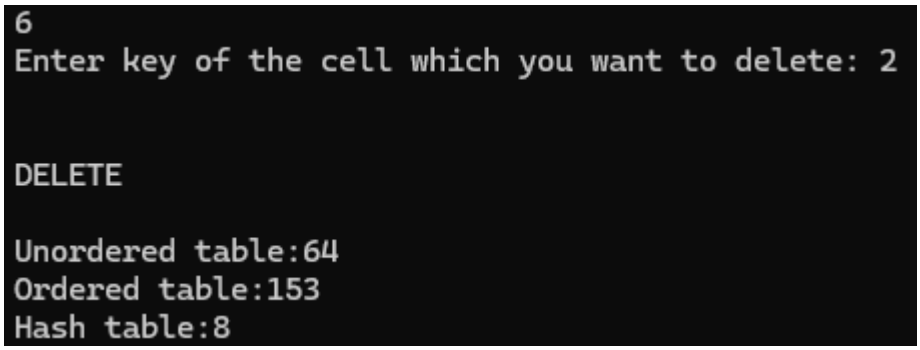
Опция 5.4 — SAVE TO TABLES - запускает сценарий вывода на консоль всех полиномов. Ожидает на вход номер полинома из списка и целочисленный или вещественный ключ. Сохраняет его в таблицу и выводит количество операций, сделанных в таблице каждого вида.

```
4
List of existing polinoms:
#1   Your polinom:  x^5 y^5 z^5
#2   Your polinom: 5 x^2 y^3 z^4
Choose number of polinom: 1
Enter key of this cell: 1
SAVE

Unordered table:1
Ordered table:6
Hash table:4
```

Опция 5.5 — FIND IN TABLES - Ожидает на вход целочисленный или вещественный ключ. Если элемент с таким ключом найден, то выводит количество операций, сделанных в таблице каждого вида, иначе выводит надпись, что такого ключа не существует.



Опция 5.6 — DELETE FROM TABLES - Ожидает на вход целочисленный или вещественный ключ. Если элемент с таким ключом найден, то выводит количество операций, сделанных в таблице каждого вида, иначе выводит надпись, что такого ключа не существует.



3. Руководство программиста

3.1. Описание структуры программы

С учётом требований к реализации целесообразным является следующее представление структуры программы:

Класс Table – шаблонный класс, полями которого будут все необходимые виды шаблонных таблиц, содержащий основные методы работы со всеми классами.

Класс OrderedTable – шаблонный класс, предоставляющий возможность работы с основными методами упорядоченной таблицы.

Класс UnorderedTable – шаблонный класс, предоставляющий возможность работы с основными методами неупорядоченной таблицы.

Класс HashTable – шаблонный класс, предоставляющий возможность работы с основными методами хеш-таблицы.

Класс Polinom - шаблонный класс, предоставляющий возможность работы с основными методами и арифметическими операциями полинома.

3.2. Описание структур данных

Структуры данных состоит из следующих компонентов:

- класс Monom – Моном — выражение вида k * z^(ix) * y^(iy) * z^(iz)
- класс Polinom – Полином — сумма мономов
- класс List – шаблонный односвязный список с фиктивной головой
- класс Node – шаблонная ячейка списка
- структура Cell – Ячейка таблицы — объект, состоящий из шаблонного ключа (уникального идентификатора ячейка) и шаблонных данных
- объект OrderedTable – шаблонная упорядоченная таблица, в которой ячейки находятся в порядке возрастания относительно ключа
- объект UnorderedTable – шаблонная неупорядоченная таблица, в которой ячейки находятся в порядке вставки в таблицу

- объект HashTable – хеш-таблица, в которой ячейки располагаются в векторе. Индекс ячейки высчитывается хеш функцией. Ячейки с данными и ключами хранятся в списке, находящемся в соответствующей ячейке вектора, и расположены в отсортированном по ключу по возрастанию порядке.
- объект Tables – составная таблица, представляющая возможность работы сразу со всеми тремя видами таблиц.

3.2.1. Описание структуры полинома

3.2.1.1. Описание структуры Node

```
template <class T>
struct Node {
    T data;
    Node<T>* next;
public:
    Node() { ... }

    Node(T _data) { ... }

    Node(T _data, Node<T>* _next) { ... }

    bool operator ==(const Node& n) { ... }

    bool operator !=(const Node& n) { ... }
};
```

Node – ячейка односвязного списка с фиктивной головой. Для структуры разработаны конструкторы по умолчанию и инициализации. Перегруженные операторы сравнения сравнивают только данные ноды, без учёта указателя на следующую ноду.

3.2.1.2. Описание класса List

```
template <class T>
class List {
    Node<T>* head;
public:
    List() { ... }

    List(const List<T>& _l) { ... }

    Node<T>* HeadOut() const { return head; };

    bool IsEmpty() { ... }

    void Push(T elem) { ... }



    void Delete(Node<T>* ptr) { ... }

    void Sort() { ... }

    bool operator ==(List<T> l) { ... }

    bool operator >(List<T>& l) { ... }

    bool operator <(List<T>& l) { ... }
};
```

List – односвязный список с фиктивной головой. Реализованы конструкторы по умолчанию и конструктор копирования. Существует функция для получения указателя на голову списка и функция для проверки списка на пустоту (сравнивает указатель на голову с нулевым указателем).

Метод Push вставляет ноду на последнее место в листе и запускает сортировку.

Метод Delete получает указатель на ноду, ищет в листе ноду с эквивалентными ей данными и перепривязывает указатель предыдущей ноды в листе к следующей ноде после той, что удаляем. Если искомой ноды в листе нет, выбрасывает исключение.

Метод Sort сначала определяет длину листа, а потом запускает усовершенствованную сортировку пузырьком по данным всех нод.

Операторы сравнения ==, > и < и заключаются в последовательном сравнении данных нод из двух списков.

3.2.1.3. Описание структуры Monom

```
struct Monom {
    double k;
    int deg;
public:
    Monom() { ... }

    Monom(double _k, int _deg) { ... }

    void operator =(const Monom& m) { ... }

    bool operator >(const Monom& m) { ... }

    bool operator <(const Monom& m) { ... }

    bool operator ==(const Monom& m) { ... }

    bool operator !=(const Monom& m) { ... }
};
```

Поле k -коэффициент монома, deg -степени переменных, причем первая разряд сотых — степень x, десятых — y, единиц — z.

Реализованы конструкторы по умолчанию и инициализации, а также перегружены операторы сравнения ==, , !=, <, > и оператор присваивания

3.2.1.4. Описание класса Polinom

```
class Polinom {
    List<Monom> poli;
public:
    Polinom() {
        poli = List<Monom>();
    }

    Polinom(const Polinom& p);

    List<Monom> PoliOut() { return poli; }

    void SearchSimilar();

    void Push(Monom monom);

    void Show();

    void DeleteZeroMonoms();

    //void Sort();

    void operator +=(Polinom& p);

    Polinom& operator =(const Polinom& _p);

    Polinom& operator =(double _k);

    Polinom& operator =(int _k);

    Polinom& operator +(Polinom& _p);

    Polinom& operator -(Polinom& _p);

    Polinom& operator *(double p);

    Polinom& operator *(Polinom& _p);

    bool operator ==(Polinom& _p);

    bool operator >(Polinom& _p);

    bool operator <(Polinom& _p);

    friend ostream& operator <<(ostream& out, Polinom p){ ... }
};
```

Непосредственно полином реализован на базе списка, где ноды хранят данные типа Monom. Для класса реализованы конструкторы по умолчанию и

копирования. Также реализованы операторы сравнения ==, <, >, арифметические операции +, -, * (перегружена для вещественных чисел и полиномов), +=, = (перегружена для целочисленных, вещественных чисел и полиномов). Также перегружен оператор вывода в консоль полинома.

Функция PoliOut – возвращает поле poli.

Метод SearchSimilar — ищет подобные мономы в полиноме путём попарного сравнения степеней двух соседних мономов. Эта операция справедлива за счёт того, что в полиноме мономы хранятся в отсортированном по степени виду.

Метод Push – добавляет в полином ноду: запускает метод Push у списка, сортирует список, удаляет нулевые мономы и ищет подобные.

Метод Show – выводит в консоль список poli.

Метод DeleteZeroMonoms – последовательно сравнивает коэффициенты монома и, если они принадлежат отрезку от -0.000001 до 0.000001, удаляет моном из списка вызовом метода Delete для списка poli.

3.2.2. Описание структур таблиц

```
template<class Tkey, class Tdata>

class Tables {
    unorderedTable<Tkey, Tdata> UnT;
    orderedTable<Tkey, Tdata> OrT;
    hashTable<Tkey, Tdata> HT;

public:

    Tables() { ... }

    Tables(int _N) { ... }

    void Push(Cell<Tkey, Tdata> cell) { ... }

    void Delete(Tkey _key) { ... }

    void Find(Tkey _key) { ... }
};
```

Класс Tables отвечает за работу сразу со всеми видами таблиц. Для класса реализован конструктор по умолчанию и конструктор инициализации аргумента хеш-функции (_N). Также существуют методы вставки, поиска и удаления, которые последовательно вызывают соответствующие методы у каждой таблицы и выводят на экран количество операций.

3.2.2.1 Неупорядоченная таблица

```
template <class Tkey, class Tdata>

class unorderedTable {
    vector<Cell<Tkey, Tdata>> table;
public:
    unorderedTable() { ... }

    vector<Cell<Tkey, Tdata>> tableOut() { ... }

    void Push(Cell<Tkey, Tdata> cell) { ... }

    int Find(Tkey _key) { ... }

    void Delete(Tkey _key) { ... }
};
```

Класс состоит из одного поля — не отсортированного вектора ячеек с информацией. Существует конструктор по умолчанию и метод tableOut, возвращающий поле table.

Метод Push – вставляет ячейку в конец вектора.

Метод Find – проходит по вектору сравнивая искомый ключ с ключом в каждой ячейке. Если ключ найден, то метод возвращает индекс ячейки. Если не найден, вызывает исключение, обрабатываемое в приложении.

Метод Delete – вызывает методы Find и, если элемент найден, заменяет его на последний элемент в векторе и удаляет последний элемент.

3.2.2.2 Упорядоченная таблица

```
template <class Tkey, class Tdata>

class orderedTable {
    vector<Cell<Tkey, Tdata>> table;
public:
    orderedTable() { ... }

    vector<Cell<Tkey, Tdata>> tableOut() { ... }

    void Push(Cell<Tkey, Tdata> cell) { ... }

    int Find(Tkey _key) { ... }

    void Delete(Tkey _key) { ... }
};
```

Класс состоит из одного поля — отсортированного вектора ячеек с информацией. Существует конструктор по умолчанию и метод tableOut, возвращающий поле table.

Метод Push – запускает алгоритм бинарного поиска по ключу ячеек. Если после завершения алгоритма левая граница равна правой границе, то отдельно рассматриваем случай, когда вектор пустой и когда не пустой. Если не пустой, то сдвигаем все элементы с с индексом больше правой границы влево и вставляем элемент. Если левая и правая границы не равны, то рассматриваем два случая: когда ключ элемента с индексом правой границы меньше ключа и когда он не меньше ключа, и в соответствии с этой информацией сдвигаем элементы и вставляем наш.

Метод Find – запускает алгоритм бинарного поиска. Если элемент найден, то возвращает индекс, если нет, вызывает исключение, обрабатываемое в приложении.

Метод Delete – вызывает метод Find и удаляет элемент по возвращаемому индексу сдвигом всех элементов правее найденного на одну ячейку влево и уменьшением длины вектора на 1.

3.2.2.3 Хеш-таблица

```
template <class Tkey, class Tdata>
class hashTable {
    int N;
    vector <List<Cell<Tkey, Tdata>>> table;
public:
    hashTable() { ... }

    hashTable(int _N) { ... }

    int Hash(Tkey key) { ... }

    int NOut() { ... }

    vector<List<Cell<Tkey, Tdata>>> tableOut() { ... }

    void Push(Cell<Tkey, Tdata> cell) { ... }

    Cell<Tkey, Tdata> Find(Tkey _key) { ... }

    void Delete(Tkey _key) { ... }
};
```

Класс состоит из двух полей — вектора списков ячеек с информацией и аргумента хеш функции. Существует конструктор по умолчанию и конструктор инициализации аргумента хеш функции, метод tableOut, возвращающий поле table, метод NOut – возвращающий аргумент хеш функции, а также метод Hash – высчитывающий значение хеш функции (в данной реализации это остаток от деления на N).

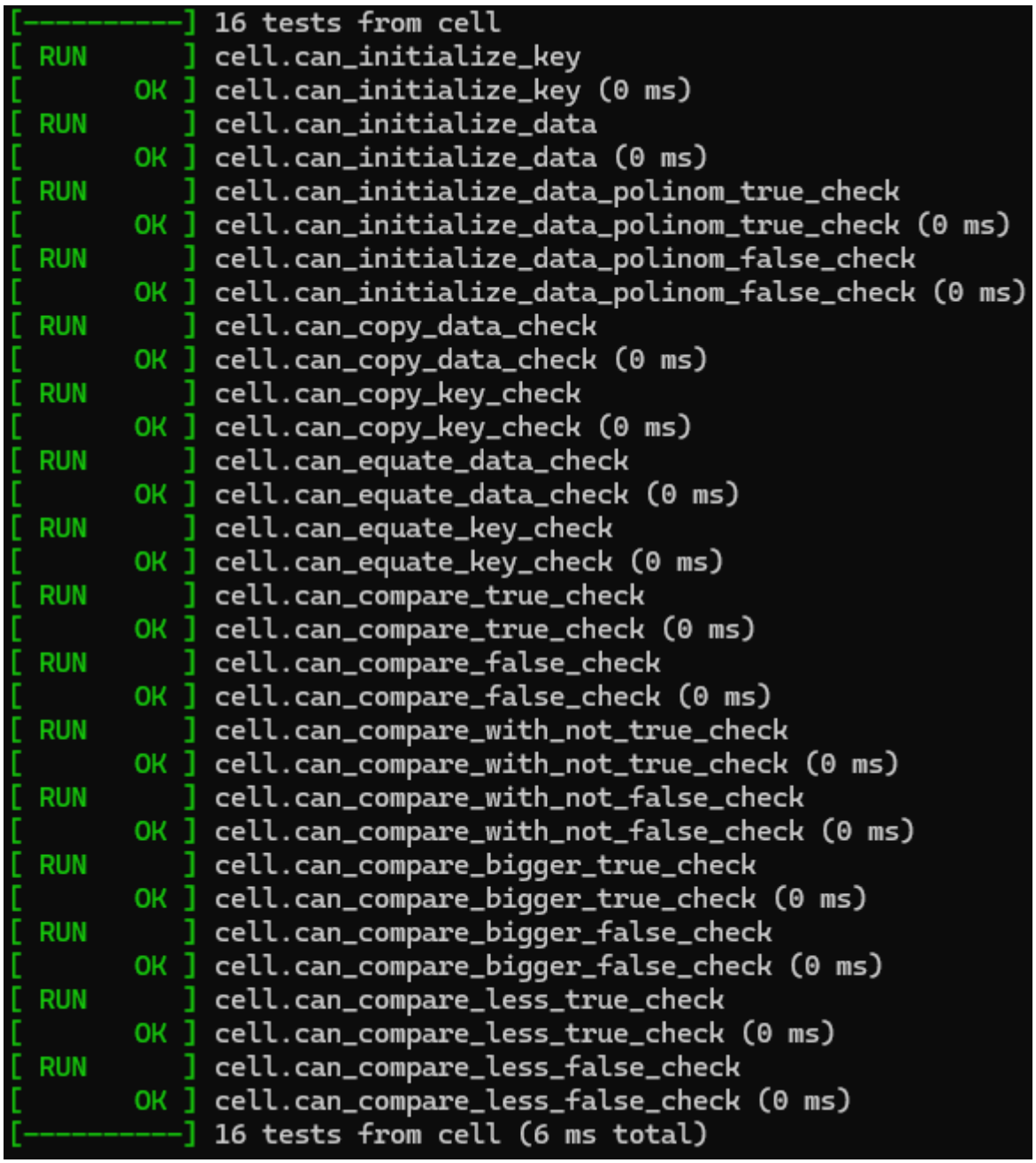
Метод Push – получает индекс вектора из метода Hash и вставляет элемент в список под этим номером посредством вызова Push для List.

Метод Find – получает индекс вектора из метода Hash и ищет нужный элемент в соответствующем списке. Возвращает ячейку с информацией.

Метод Delete – получает индекс вектора из метода Hash, ищет нужный элемент в соответствующем списке и переприязывает указатель предыдущего элемента к следующему, после удаляемого.

4. Проверка корректности

С целью контроля правильности работы структур данных были написаны тесты с использованием gtest для структуры Cell и классов unorderedTable, orderedTable и HashTable.



```
[----------] 9 tests from hashTable
[ RUN      ] hashTable.can_initialize_N
[       OK ] hashTable.can_initialize_N (0 ms)
[ RUN      ] hashTable.can_initialize_data
4[       OK ] hashTable.can_initialize_data (0 ms)
[ RUN      ] hashTable.can_count_hash
[       OK ] hashTable.can_count_hash (0 ms)
[ RUN      ] hashTable.can_push_true_check
44[       OK ] hashTable.can_push_true_check (0 ms)
[ RUN      ] hashTable.can_push_false_check
44[       OK ] hashTable.can_push_false_check (0 ms)
[ RUN      ] hashTable.can_find_true_check
4412[       OK ] hashTable.can_find_true_check (1 ms)
[ RUN      ] hashTable.can_find_false_check
412[       OK ] hashTable.can_find_false_check (0 ms)
[ RUN      ] hashTable.can_delete_true_check
44812[       OK ] hashTable.can_delete_true_check (1 ms)
[ RUN      ] hashTable.can_delete_false_check
412[       OK ] hashTable.can_delete_false_check (0 ms)
[----------] 9 tests from hashTable (4 ms total)

[----------] 6 tests from orderedTable
[ RUN      ] orderedTable.can_create
[       OK ] orderedTable.can_create (0 ms)
[ RUN      ] orderedTable.can_push
61012191934[       OK ] orderedTable.can_push (1 ms)
[ RUN      ] orderedTable.can_find_true_check
6101219193418[       OK ] orderedTable.can_find_true_check (0 ms)
[ RUN      ] orderedTable.can_find_false_check
6101219193424[       OK ] orderedTable.can_find_false_check (1 ms)
[ RUN      ] orderedTable.can_delete_true_check
61012191934188[       OK ] orderedTable.can_delete_true_check (0 ms)
[ RUN      ] orderedTable.can_delete_false_check
6101219193424[       OK ] orderedTable.can_delete_false_check (0 ms)
[----------] 6 tests from orderedTable (3 ms total)

[----------] 6 tests from unorderedTable
[ RUN      ] unorderedTable.can_create
[       OK ] unorderedTable.can_create (0 ms)
[ RUN      ] unorderedTable.can_push
11[       OK ] unorderedTable.can_push (0 ms)
[ RUN      ] unorderedTable.can_find_true_check
117[       OK ] unorderedTable.can_find_true_check (0 ms)
[ RUN      ] unorderedTable.can_find_false_check
11[       OK ] unorderedTable.can_find_false_check (0 ms)
[ RUN      ] unorderedTable.can_delete_true_check
1174[       OK ] unorderedTable.can_delete_true_check (1 ms)
[ RUN      ] unorderedTable.can_delete_false_check
1[       OK ] unorderedTable.can_delete_false_check (0 ms)
[----------] 6 tests from unorderedTable (2 ms total)
```

Заключение

В ходе написания лабораторной работы было разработано расширение для статической библиотеки Polinom, позволяющее хранить и обрабатывать объекты типа Polinom в таблицах 3-х видов: упорядоченной, неупорядоченной и хеш-таблице. Благодаря алгоритмам написания основных методов стало возможным сравнивать сложность действий Вставка, Поиск и Удаление в таблицах разных видов. Также сохранилась возможность совершать арифметические действия с полиномами и с целыми или вещественными числами.

Список литературы

1. Документация по языку — Режим доступа: C++https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=msvc-170

2. Хеш-тфблицы — Режим доступа: https://habr.com/ru/articles/509220/

Приложение

Приложение 1. list.h

```
template <class T>
struct Node {
	T data;
	Node<T>* next;
public:
	Node() {
		next = nullptr;
	}

	Node(T _data) {
		data = _data;
		next = nullptr;
	}

	Node(T _data, Node<T>* _next) {
		data = _data;
		next = _next;
	}

	bool operator ==(const Node& n) {
		if (data == n.data)
			return true;
		return false;
	}

	bool operator !=(const Node& n) {
		return !(this == n);
	}
};

template <class T>
class List {
	Node<T>* head;
public:
	List() {
		head = new Node<T>;
		head->next = nullptr;
	}
```

```
List(const List<T>& _l) {
	Node<T>* tmp_head = new Node<T>;
	head = tmp_head;
	Node<T>* tmp_p_head = _l.head;
	while (tmp_p_head->next != nullptr) {
		tmp_head->next = new Node<T>(tmp_p_head->next->data);
		tmp_head = tmp_head->next;
		tmp_p_head = tmp_p_head->next;
	}
}

Node<T>* HeadOut() const { return head; };

bool IsEmpty() {
	return (head -> next == nullptr);
}

void Push(T elem) {
	if (IsEmpty())
		head->next = new Node<T>(elem, nullptr);
	else
	{
		Node<T>* tmp = head -> next;
		while (tmp->next != nullptr)
			tmp = tmp->next;
		tmp->next = new Node<T>(elem, nullptr);
	}
	Sort();
}



void Delete(Node<T>* ptr) {
	Node<T>* tmp = head;
	while ((tmp->next != ptr) && (tmp->next != nullptr)){
		tmp = tmp->next;
	}
	if ((tmp->next != nullptr) && (tmp->next == ptr))
		tmp->next = tmp->next->next;
	else
		throw "there_are_no_elements_to_delete";
}
```

```
void Sort() {
	Node<T>* tmp_head = head;
	Node<T>* tmp_swap = head; // empty node for swapping
	Node<T>* tmp_prev = head; // storage for pointer to tmp_head
	Node<T>* tmp = head; // next node after tmp_head
	int len = 0, f = 1, j, i = 0;

	while (tmp_head->next != nullptr) {
		len++;
		tmp_head = tmp_head->next;
	}
	while (i < len && f == 1) {
		tmp_prev = head;
		tmp_head = head->next;
		tmp = tmp_head->next;
		j = len - i - 1;
		while (j > 0) {
			if (tmp_head->data < tmp->data) { // swapping
				tmp_swap = tmp->next;
				tmp->next = tmp_head;
				tmp_prev->next = tmp;
				tmp_head->next = tmp_swap;
				f = 1;
				tmp_swap = tmp;
				tmp = tmp_head;
				tmp_head = tmp_swap;

			}
			tmp = tmp->next;
			tmp_head = tmp_head->next;
			tmp_prev = tmp_prev->next;
			j--;
		}
		i++;
	}
}

bool operator ==(List<T> l) {
	Node<T>* tmp = head;
	Node<T>* tmp_l = l.head;
	while (tmp->next != nullptr) {
```

```
                if ((tmp_l->next == nullptr)||(tmp->data != tmp_l->data))
                        return false;
                tmp = tmp->next;
                tmp_l = tmp_l->next;
        }
        if ((tmp_l->next != nullptr) || (tmp->data != tmp_l->data))
                return false;
        return true;
    }

    bool operator >(List<T>& l) {
        Node<T>* tmp = head;
        Node<T>* tmp_l = l.head;
        while (tmp->next != nullptr) {
                if ((tmp_l->next == nullptr) || (tmp->data > tmp_l->data))
                        return true;
                tmp = tmp->next;
                tmp_l = tmp_l->next;
        }
        if ((tmp_l->next != nullptr) || !(tmp->data > tmp_l->data))
                return false;
        return true;
    }

    bool operator <(List<T>& l) {
        Node<T>* tmp = head;
        Node<T>* tmp_l = l.head;
        while (tmp->next != nullptr) {
                if ((tmp_l->next == nullptr) || (tmp->data > tmp_l->data))
                        return false;
                tmp = tmp->next;
                tmp_l = tmp_l->next;
        }
        if ((tmp_l->next != nullptr) || (tmp->data < tmp_l->data))
                return true;
        return false;
    }
};
```

Приложение 2.1. polinom.h

```
#include "list.h"
#include <iostream>
```

```
using namespace std;

struct Monom {
	double k;
	int deg;
public:
	Monom() {
		k = 0;
		deg = 0;
	}

	Monom(double _k, int _deg) {
		if (_deg > 999 || _deg < 0)
			throw "deg_is_out_of_range";
		k = _k;
		deg = _deg;
	}

	void operator =(const Monom& m) {
		k = m.k;
		deg = m.deg;
	}

	bool operator >(const Monom& m) {
		if ((deg > m.deg)||(deg == m.deg && k > m.k))
			return true;
		return false;
	}

	bool operator <(const Monom& m) {
		if ((deg < m.deg) || (deg == m.deg && k < m.k))
			return true;
		return false;
	}

	bool operator ==(const Monom& m) {
		if ((deg == m.deg) && (k == m.k))
			return true;
		return false;
	}
```

```
	bool operator !=(const Monom& m) {
		return !(*this == m);
	}
};

class Polinom {
	List<Monom> poli;
public:
	Polinom() {
		poli = List<Monom>();
	}

	Polinom(const Polinom& p);

	List<Monom> PoliOut() { return poli; }

	void SearchSimilar();

	void Push(Monom monom);

	void Show();

	void DeleteZeroMonoms();

	//void Sort();

	void operator +=(Polinom& p);

	Polinom& operator =(const Polinom& _p);

	Polinom& operator =(double _k);

	Polinom& operator =(int _k);

	Polinom& operator +(Polinom& _p);

	Polinom& operator -(Polinom& _p);

	Polinom& operator *(double p);

	Polinom& operator *(Polinom& _p);
```

```
	bool operator ==(Polinom& _p);

	bool operator >(Polinom& _p);

	bool operator <(Polinom& _p);

	friend ostream& operator <<(ostream& out, Polinom p) {
		Node<Monom>* tmp = p.PoliOut().HeadOut();
		while (tmp->next != nullptr) {
			if (tmp == p.PoliOut().HeadOut()) {
				if (tmp->next->data.k != 1 && tmp->next->data.k != -1 || tmp-
>next->data.deg == 0)
					out << tmp->next->data.k;
				else {
					if (tmp->next->data.k == -1)
						out << "-";
				}
				if (tmp->next->data.deg / 100 != 0)
					out << " x^" << tmp->next->data.deg / 100;
				if (tmp->next->data.deg / 10 % 10 != 0)
					out << " y^" << tmp->next->data.deg / 10 % 10;
				if (tmp->next->data.deg % 100 != 0)
					out << " z^" << tmp->next->data.deg % 10;
			}
			else {
				if (tmp->next->data.k >= 0)
					out << " + ";
				else
					out << " - ";
				if (tmp->next->data.k != 1 && tmp->next->data.k != -1)
					out << abs(tmp->next->data.k);
				if (tmp->next->data.deg / 100 != 0)
					out << " x^" << tmp->next->data.deg / 100;
				if (tmp->next->data.deg / 10 % 10 != 0)
					out << " y^" << tmp->next->data.deg / 10 % 10;
				if (tmp->next->data.deg % 100 != 0)
					out << " z^" << tmp->next->data.deg % 10;
			}
			tmp = tmp->next;
		}
		return out;
	};
```

```
};
```

Приложение 2.2. polinom.cpp

```
Polinom::Polinom(const Polinom& p) {
	Node<Monom>* tmp_head = poli.HeadOut();
	Node<Monom>* tmp_p_head = p.poli.HeadOut();
	while (tmp_p_head->next != nullptr) {
		tmp_head->next = new Node<Monom>(tmp_p_head->next->data);
		tmp_head = tmp_head->next;
		tmp_p_head = tmp_p_head->next;
	}
}
void Polinom::SearchSimilar() {
	Node<Monom>* tmp_head = poli.HeadOut();
	while (tmp_head->next != nullptr && tmp_head->next->next != nullptr) {
		if (tmp_head->next->data.deg == tmp_head->next->next->data.deg) {
			tmp_head->next->data.k = tmp_head->next->data.k + tmp_head->next-
>next->data.k;
			poli.Delete(tmp_head->next->next);
		}
		else
			tmp_head = tmp_head->next;
	}
	DeleteZeroMonoms();
}

void Polinom::Push(Monom monom) {
	poli.Push(monom);
	poli.Sort();
	DeleteZeroMonoms();
	SearchSimilar();
}

void Polinom::Show() {
	Node<Monom>* tmp = poli.HeadOut();
	cout << "Your polinom: ";
	while (tmp->next != nullptr) {
		if (tmp == poli.HeadOut()) {
			if (tmp->next->data.k != 1 && tmp->next->data.k != -1 || tmp->next-
>data.deg == 0)
				cout << tmp->next->data.k;
```

```
                else {
                        if (tmp->next->data.k == -1)
                                cout << "-";
                }
                if(tmp->next->data.deg / 100 != 0)
                        cout << " x^" << tmp->next->data.deg / 100;
                if (tmp->next->data.deg / 10 % 10 != 0)
                        cout << " y^" << tmp->next->data.deg / 10 % 10;
                if (tmp->next->data.deg % 100 != 0)
                        cout << " z^" << tmp->next->data.deg % 10;
        }
        else {
                if (tmp->next->data.k >= 0)
                        cout << " + ";
                else
                        cout << " - ";
                if(tmp->next->data.k != 1 && tmp->next->data.k != -1)
                        cout << abs(tmp->next->data.k);
                if (tmp->next->data.deg / 100 != 0)
                        cout << " x^" << tmp->next->data.deg / 100;
                if (tmp->next->data.deg / 10 % 10 != 0)
                        cout << " y^" << tmp->next->data.deg / 10 % 10;
                if (tmp->next->data.deg % 100 != 0)
                        cout << " z^" << tmp->next->data.deg % 10;
        }
        tmp = tmp->next;
    }
}

void Polinom::DeleteZeroMonoms() {
    Node<Monom>* tmp_head = poli.HeadOut();
    while (tmp_head->next != nullptr) {
        if ((tmp_head->next->data.k < 0.000001) && (tmp_head->next->data.k > -
0.000001))
            poli.Delete(tmp_head->next);
        else
            tmp_head = tmp_head->next;
    }
}

void Polinom::operator +=(Polinom& p) {
    Node<Monom>* tmp_head = poli.HeadOut();
    Node<Monom>* tmp_p_head = p.poli.HeadOut();
```

```
	Node<Monom>* tmp = poli.HeadOut(); // storage for temporary pointer
	while (tmp_head->next != nullptr && tmp_p_head->next != nullptr) {
		if (tmp_head->next != nullptr && tmp_p_head->next != nullptr &&
(tmp_head->next->data.deg == tmp_p_head->next->data.deg)) {
			tmp_head->next->data.k = tmp_head->next->data.k + tmp_p_head->next-
>data.k;
			tmp_head = tmp_head->next;
			tmp_p_head = tmp_p_head->next;
		}
		else
			if (tmp_head->next->data.deg > tmp_p_head->next->data.deg) {
				tmp_head = tmp_head->next;
			}
			else {
				tmp = tmp_head->next;
				tmp_head->next = new Node<Monom>(tmp_p_head->next->data,
tmp_head->next);
				tmp_p_head = tmp_p_head->next;
				tmp_head = tmp_head->next;
			}
	}
	if(tmp_p_head->next != nullptr)
		while (tmp_p_head->next != nullptr) {
			tmp_head->next = new Node<Monom>(tmp_p_head->next->data);
			tmp_p_head = tmp_p_head->next;
			tmp_head = tmp_head->next;
		}
	DeleteZeroMonoms();
}

Polinom& Polinom::operator =(const Polinom& _p) {
	Polinom* result = new Polinom(_p);
	poli = result->poli;
	return *this;
}

Polinom& Polinom::operator =(int _k) {
	Monom M((double)_k, 0);
	Polinom* result = new Polinom();
	result->Push(M);
	poli = result->poli;
```

```
	return *this;
}

Polinom& Polinom::operator =(double _k) {
	Monom M(_k, 0);
	Polinom* result = new Polinom();
	result->Push(M);
	poli = result->poli;

	return *this;
}


Polinom& Polinom::operator +(Polinom& _p) {
	Polinom* result = new Polinom(*this);
	*result += _p;
	return *result;
}

Polinom& Polinom::operator *(double p) {
	Polinom* result = new Polinom(*this);
	Node<Monom>* tmp_head = result->poli.HeadOut();
	while (tmp_head->next != nullptr) {
		tmp_head->next->data.k = tmp_head->next->data.k * p;
		tmp_head = tmp_head->next;
	}
	result->DeleteZeroMonoms();
	return *result;
}

Polinom& Polinom::operator -(Polinom& _p) {
	Polinom* result = new Polinom(*this);
	*result += _p * -1;
	return *result;
}

Polinom& Polinom::operator *(Polinom& _p) {
	Polinom* result = new Polinom();
	Node<Monom>* tmp_head = this->poli.HeadOut();
	Node<Monom>* tmp_p_head = _p.poli.HeadOut();
	Monom* new_node;
	while (tmp_head->next != nullptr) {
```

```
			Node<Monom>* tmp_p_head = _p.poli.HeadOut();
			while (tmp_p_head->next != nullptr) {
				new_node = new Monom(tmp_head->next->data.k * tmp_p_head->next-
>data.k, tmp_head->next->data.deg + tmp_p_head->next->data.deg);
				result->Push(*new_node);
				tmp_p_head = tmp_p_head->next;
			}
			tmp_head = tmp_head->next;
		}
		result->DeleteZeroMonoms();
		return *result;
}

bool Polinom::operator ==(Polinom& _p) {
		return (poli == _p.poli);
}

bool Polinom::operator >(Polinom& _p) {
		return (poli > _p.poli);
}

bool Polinom::operator <(Polinom& _p) {
		return (poli < _p.poli);;
}
```

Приложение 3. cell.h

```
template <class Tkey, class Tdata>
struct Cell {
		Tkey key;
		Tdata data;
public:
		Cell() {
			key = 0;
			data = 0;
		}

		Cell(Tkey _key) {
			key = _key;
		}

		Cell(Tkey _key, Tdata _data) {
```

```
	key = _key;
	data = _data;
}

Cell(const Cell<Tkey, Tdata>& p) {
	key = p.key;
	data = p.data;
}

void Show() {
		cout << "Key: " << key << endl << "Data: " << data;
}

Tkey keyOut() {
	return key;
}

Tdata dataOut() {
	return data;
}

Cell<Tkey, Tdata>& operator =(Cell<Tkey, Tdata>& p) {
	key = p.key;
	data = p.data;
	return *this;
}

bool operator ==(Cell<Tkey, Tdata>& p) {
	if ((key == p.key)&&(data == p.data))
		return true;
	else
		return false;
}

bool operator !=(Cell<Tkey, Tdata>& p) {
	return !(*this == p);
}

bool operator >(Cell<Tkey, Tdata>& p) {
	return (data > p.data);
}
```

```
	bool operator <(Cell<Tkey, Tdata>& p) {
		return (data < p.data);
	}
};
```

Приложение 4. unorderdTable.h

```
#pragma once
#include <iostream>
#include <vector>
#include "cell.h"
#include "polinom.h"

template <class Tkey, class Tdata>

class unorderedTable {
	vector<Cell<Tkey, Tdata>> table;
public:
	unorderedTable() {
		table = vector<Cell<Tkey, Tdata>>();
	}

	vector<Cell<Tkey, Tdata>> tableOut() {
		return table;
	}

	void Push(Cell<Tkey, Tdata> cell) {
		int COUNTER = 0;

		table.push_back(cell);
		COUNTER++;

		cout << COUNTER;
	}

	int Find(Tkey _key) {
		int COUNTER = 0;

		int i = 0;
		COUNTER ++;
		while ((i < table.size()) && (table[i].keyOut() != _key)) {
			COUNTER += 3;
```

```
                i++;
            }
            COUNTER += 2;
            if (i == table.size()) {
                throw "Cell isn't found";
            }
            else {
                COUNTER++;
                cout << COUNTER;
                return i;
            }
        }

        void Delete(Tkey _key) {
            int COUNTER = 0;

            int index = Find(_key);
            COUNTER++;
            table[index] = table[table.size() - 1];
            table.pop_back();
            COUNTER += 3;
            cout << COUNTER;
        }
};
```

Приложение 5. orderdTable.h

```
#include <iostream>
#include <vector>
#include "cell.h"
#include "polinom.h"
#include <algorithm>

template <class Tkey, class Tdata>

class orderedTable {
        vector<Cell<Tkey, Tdata>> table;
public:
        orderedTable() {
            table = vector<Cell<Tkey, Tdata>>();
        }
```

```
vector<Cell<Tkey, Tdata>> tableOut() {
	return table;
}

void Push(Cell<Tkey, Tdata> cell) {
	int COUNTER = 0;

	int key = cell.keyOut();
	COUNTER++;
	int l = 0, r = max((int)(table.size() - 1), 0);
	COUNTER += 2;
	while (r - l > 1) {
		int i = (l + r) / 2;
		COUNTER += 5;
		if (table[i].keyOut() < key) {
			l = i + 1;
			COUNTER += 4;
		}
		else
			if (table[i].keyOut() > key) {
				r = i - 1;
				COUNTER += 4;
			}
	}
	if (r == l) {
		COUNTER++;
		if (table.size() == 0) {
			table.push_back(cell);
			COUNTER += 2;
		}
		else{
			table.push_back(table[table.size() - 1]);
			COUNTER += 3;
			if(table[r].keyOut() < key){
				COUNTER += 2;
				for (int i = table.size() - 2; i > r; i--) {
					table[i] = table[i - 1];
					COUNTER += 5;
				}
				table[r+1] = cell;
				COUNTER++;
				}
```

```
                        else{
                                for (int i = table.size() - 2; i >= r; i--) {
                                        table[i] = table[i - 1];
                                        COUNTER += 5;
                                }
                                table[r] = cell;
                                COUNTER++;
                                }
                        }
        }
        else {
                table.push_back(table[table.size() - 1]);
                COUNTER += 3;
                if (table[r].keyOut() < key) {
                        COUNTER += 3;
                        for (int i = table.size() - 2; i > r; i--) {
                                table[i] = table[i - 1];
                                COUNTER += 5;
                        }
                        table[r + 1] = cell;
                        COUNTER++;
                        }
                else{
                        for (int i = table.size() - 2; i >= r; i--) {
                                table[i] = table[i - 1];
                                COUNTER += 5;

                        }
                        table[r] = cell;
                        COUNTER++;
                        }
                }
        cout << COUNTER;
}

int Find(Tkey _key) {
        int COUNTER = 0;

        int l = 0, r = max((int)(table.size() - 1), 0);
        COUNTER += 4;
        while (r > l) {
                COUNTER += 2;
```

```
            int i = (l + r) / 2;
            COUNTER += 3;
            if (table[i].keyOut() < _key) {
                  l = i + 1;
                  COUNTER += 4;
            }
            else
                  if (table[i].keyOut() > _key) {
                        r = i - 1;
                        COUNTER += 4;
                  }
                  else {
                        cout << COUNTER;
                        return i;
                  }
      }

      if ((table.size() != 0)&&(table[l].keyOut() == _key)) {
            COUNTER += 2;
            cout << COUNTER;
            return l;
      }
      else {
            COUNTER += 2;
            cout << COUNTER;
            throw "Cell isn't found";
      }
}

void Delete(Tkey _key){
      int COUNTER = 0;
      int index = Find(_key);
      COUNTER++;
      COUNTER++;
      for (int i = index; i < table.size() - 1; i++) {
            table[i] = table[i+1];
            COUNTER += 5;
      }
      table.pop_back();
      COUNTER++;
      cout << COUNTER;
}
```

```
};
```

Приложение 6. hashTable.h

```
#include <iostream>
#include <vector>
#include "cell.h"
#include "polinom.h"

using namespace std;

template <class Tkey, class Tdata>
class hashTable {
	int N;
	vector <List<Cell<Tkey, Tdata>>> table;
public:
	hashTable() {
		N = 1;
		table = vector<List<Cell<Tkey, Tdata>>>(N);
	}

	hashTable(int _N) {
		N = _N;
		table = vector<List<Cell<Tkey, Tdata>>>(N);
	}

	int Hash(Tkey key) {
		return ((int)key % N);
	}

	int NOut() {
		return N;
	}

	vector<List<Cell<Tkey, Tdata>>> tableOut() {
		return table;
	}

	void Push(Cell<Tkey, Tdata> cell) {
		int COUNTER = 0;
		int index = Hash(cell.key);
		table[index].Push(cell);
```

```
		COUNTER += 4;
		cout << COUNTER;
	}

	Cell<Tkey, Tdata> Find(Tkey _key) {
		int COUNTER = 0;
		int index = Hash(_key);
		COUNTER += 2;
		Node<Cell<Tkey, Tdata>>* tmp_head = table[index].HeadOut();
		COUNTER += 3;
		while ((tmp_head->next != nullptr) && (tmp_head->next->data.keyOut() !=
_key)) {
			tmp_head = tmp_head->next;
			COUNTER += 6;
		}
		if (tmp_head->next == nullptr) {
			COUNTER++;
			cout << COUNTER;
			throw "Cell isn't found";
		}
		else {
			COUNTER++;
			cout << COUNTER;
			return tmp_head->next->data;
		}
	}

	void Delete(Tkey _key) {
		int COUNTER = 0;
		int index = Hash(_key);
		COUNTER += 2;
		Node<Cell<Tkey, Tdata>>* tmp_head = table[index].HeadOut();
		COUNTER += 3;
		while ((tmp_head->next != nullptr) && (tmp_head->next->data.keyOut() !=
_key)) {
			tmp_head = tmp_head->next;
			COUNTER += 6;
		}
		if (tmp_head->next == nullptr) {
			COUNTER++;
			cout << COUNTER;
			throw "Cell isn't found";
```

```
            }
            else {
                  COUNTER+=3;
                  cout << COUNTER;
                  tmp_head->next = tmp_head->next->next;
            }
      }
};
```

Приложение 7. tables.h

```
#include <iostream>
#include "list.h"
#include "polinom.h"
#include "unorderedTable.h"
#include "orderedTable.h"
#include "hashTable.h"

using namespace std;

template<class Tkey, class Tdata>

class Tables {
      unorderedTable<Tkey, Tdata> UnT;
      orderedTable<Tkey, Tdata> OrT;
      hashTable<Tkey, Tdata> HT;

public:

      Tables() {
            UnT = unorderedTable<Tkey, Tdata>();
            OrT = orderedTable<Tkey, Tdata>();
            HT = hashTable<Tkey, Tdata>();
      };

      Tables(int _N) {
            UnT = unorderedTable<Tkey, Tdata>();
            OrT = orderedTable<Tkey, Tdata>();
            HT = hashTable<Tkey, Tdata>(_N);
      };

      void Push(Cell<Tkey, Tdata> cell) {
```

```
            cout << "SAVE" << endl;
            cout << endl << "Unordered table:";
            UnT.Push(cell);
            cout << endl << "Ordered table:";
            OrT.Push(cell);
            cout << endl << "Hash table:";
            HT.Push(cell);
        }

        void Delete(Tkey _key) {
            cout << endl << endl << "DELETE" << endl;
            cout << endl << "Unordered table:";
            UnT.Delete(_key);
            cout << endl << "Ordered table:";
            OrT.Delete(_key);
            cout << endl << "Hash table:";
            HT.Delete(_key);
        }

        void Find(Tkey _key) {
            cout << endl << endl << "FIND" << endl;
            cout << endl << "Unordered table:";
            UnT.Find(_key);
            cout << endl << "Ordered table:";
            OrT.Find(_key);
            cout << endl << "Hash table:";
            HT.Find(_key);
        }
};
```