Федеральное государственное образовательное бюджетное учреждение

высшего профессионального образования

«Нижегородский Государственный Университет им.

Н.И. Лобачевского» (ННГУ)

Институт Информационных Технологий Математики и Механики

Отчёт по лабораторной работе

**Алгебра полиномов.**

Выполнили:

студенты группы 3821Б1ПМ3

Одинец В.В.

Мерзляков В.А.

Мезенцев Д.А.

Кожин С.В.

Качалин А.Г.

Проверил:

заведующий лабораторией суперкомпьютерных технологий и высокопроизводительных вычислений

Лебедев И. Г.

Нижний Новгород

2023г.

Содержание

1. Введение 2

2. Постановка задачи 3

3. Руководство пользователя 4

4. Руководство программиста 6

*4.1 Описание структуры программы* 6

*4.2 Описание структуры данных* 7

*4.3 Описание алгоритмов* 8

5. Эксперименты 9

6. Заключение 12

7. Литература 13

8. Приложения 14

**1. Введение.**

Алгебра полиномов – это раздел алгебры, изучающий многочлены, которые являются одним из основных объектов алгебры. Многочлены являются алгебраическими выражениями, состоящими из переменных, называемых также неизвестными, и их степеней, умноженных на коэффициенты.

Цель данной лабораторной работы – изучение основных понятий и операций, связанных с многочленами, таких как сложение, вычитание, умножение, деление многочленов. В процессе выполнения лабораторной работы студенты будут ознакомлены с основными правилами и методами работы с многочленами, решением уравнений.

Существуют задачи, в которых есть необходимость использовать табличные структуры данных. Их характерные черты состоят в том, что они хранят данные в виде пары «ключ-значение» - записи в таблице, позволяют выполнять вставку записей, поиск и удаление по ключу. Как правило, ключ предполагается уникальным для каждой записи.

Способы представления таблиц бывают разные и определяются характером решаемой задачи – например, где-то поиск происходит намного чаще, чем вставка и удаление, а где-то наоборот. В лабораторной работе рассматриваются реализации неупорядоченной, упорядоченной и хеш-таблиц.

**2. Постановка задачи.**

Требуется разработать программу, которая предоставляет возможность хранить данные в таблицах 4-х типов:

1)Неупорядоченная таблица

2)Упорядоченная таблица

3)Дерево

4)Хэш-таблица

Работа происходит сразу с таблицами всех типов.

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

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

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

Во время работы программы происходит логистика количества произведенных операций. Их число для каждой таблицы выводится на экран или в файл после выполнения соответствующей операции.

**3. Руководство пользователя.**

При запуске программы выводятся правила использования программы и корректного ввода данных.

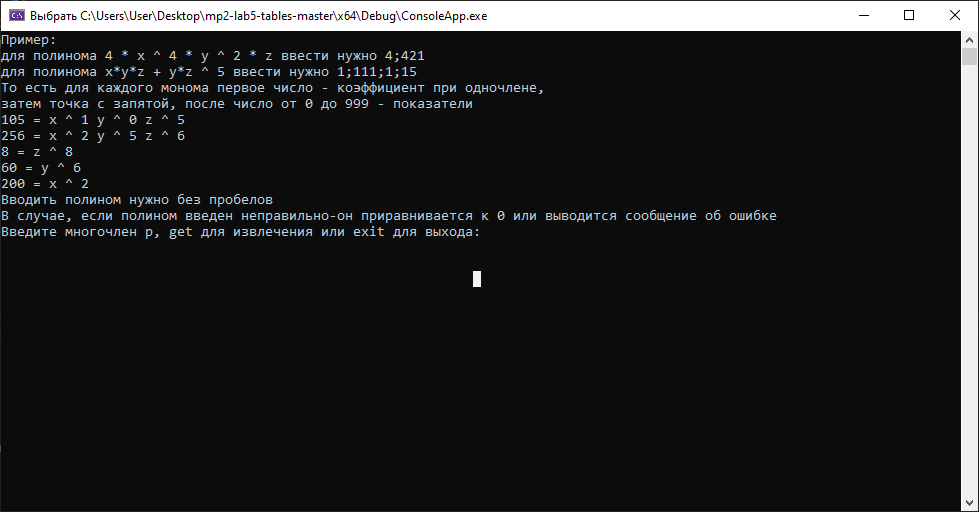


Рис.1 Консоль пользователя

Вы можете ввести «exit», чтобы закрыть программу, «get», чтобы получить многочлен, находящийся в таблице, или сам многочлен – правила его ввода отображаются при запуске программы.

При выборе варианта «get» введите имя многочлена в таблице. Если он есть, выведется его значения, и можно будет выполнить операции, если нет – вы увидите сообщение об ошибке.

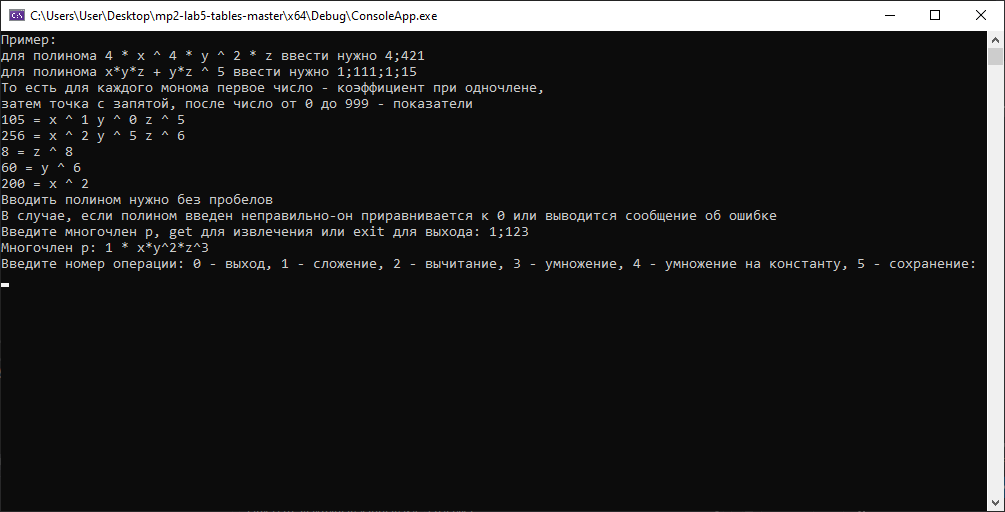


Рис.2 Консоль пользователя

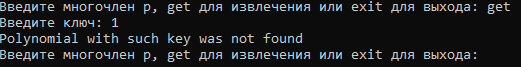


Рис.3 Консоль пользователя

Если ввод некорректен, произойдёт возврат к выбору действия.

Когда многочлен получен, можно выполнить следующие действия: выйти и программы, сложить, вычесть или умножить на многочлен, умножить на число или сохранить многочлен.

Для сохранения нужно будет ввести ключ - имя многочлена. Оно должно быть уникальным, т.е. в таблице не должен содержаться многочлен с таким же именем. Если же он есть, нужно снова ввести имя, и так до тех пор, пока оно не будет уникальным.

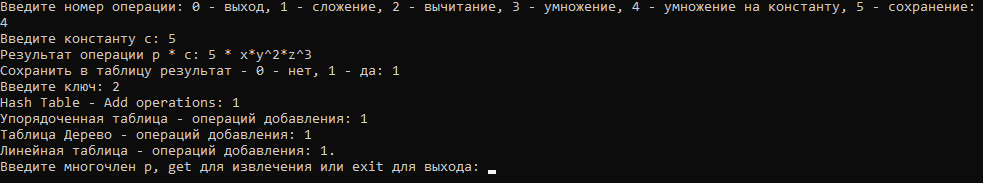


Рис.4 Консоль пользователя

Для выполнения сложения, вычитания или умножения на многочлен нужно ввести многочлен, для умножения на число – число. Ввод многочлена происходит так же, как в начале работы программы, ввод числа происходит в соответствии с форматом ввода вещественных чисел в C++.

Как только арифметическая операция выполнится, выведется её результат. Полученный многочлен можно добавить или не добавить в таблицу.

Если выбран первый вариант, требуется дать имя уникальное имя многочлену.

Далее происходит возврат к исходному окну, где вы можете выйти, взять многочлен из таблицы или ввести его вручную.

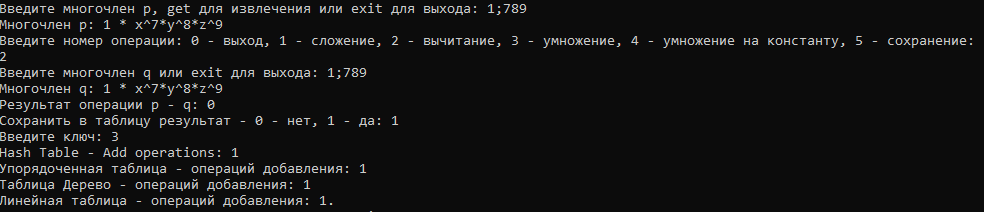


Рис.5 Консоль пользователя

**4. Руководство программиста.**

***4.1 Описание структуры программы.***

**Библиотеки, использующиеся в программе:**

1. iostream (Для ввода и вывода информации)

2. vector (Для временного хранения слагаемых многочлена)

3. sstream (Для удобной работы со строками)

4. utility (Содержит различные инструменты C++)

***4.2 Описание структуры данных.***

**Классы:**

**TableElement** – запись в виде пары «ключ-значение», является элементом, хранящимся во всех таблицах.

**LineTable** – реализация неупорядоченной таблицы с помощью std::vector.

**OrderedTable** – реализация упорядоченной таблицы с помощью двоичного дерева поиска.

**TreeNode** – вершина двоичного дерева поиска.

**TreeTable** – реализация двоичного дерева поиска.

**HashTable** – реализация хеш-таблицы с использованием метода цепочек.

**HashNode** – звено списка, хранящегося в каждой ячейке массива в хеш-таблице.

**List –** реализация списка для хранения данных.

**Monomial –** реализация мономов, использующихся в полиномах.

**Polynomial** – реализация класса полиномов.

**AVLTree –** реализация класса АВЛ-Дерева для хранения данных.

***4.3 Описание алгоритмов программы.***

**1. Пользователь может ввести слова «exit», «get» или многочлен по правилам, выводимым на экран при начале работы программы.**

**2. Ввод «exit» приводит к закрытию программы.**

**3. Ввод «get» позволяет получить многочлен из таблицы по ключу. Пользователь должен ввести ключ – имя многочлена. По ключу производится поиск в таблице; если запись с ним есть, в консоль выводятся многочлен и количество операций при поиске в каждой таблице, дальнейшие действия – на шаге 5, иначе выводится сообщение, что запись не найдена, и происходит возврат к шагу 1.**

**4. Если непосредственно вводить многочлен, при корректном вводе происходит его вывод в консоль и переход к следующему шагу, иначе – вывод сообщения об ошибке и возврат к шагу 1.**

**5. На этом шаге можно выйти из программы, выполнить арифметические операции над многочленом или сохранить его в таблицу.**

**При выборе сохранения нужно ввести уникальное имя. Если оно неуникально, придётся повторять попытку, пока имя не будет уникальным. Как только корректное имя введено, запись добавляется в таблицы, выводится количество операций при добавлении, происходит переход к шагу 1.**

**При выборе сложения, вычитания или умножения на многочлен нужно ввести второй многочлен аналогично шагу 1.**

**При выборе умножения на константу нужно ввести число. Процесс будет повторяться, пока ввод не будет корректен.**

**При выборе выхода из программы происходит завершение её работы.**

**6. По выполнении арифметических операций выводится результат и предлагается сохранить его в таблицу. Если решено сохранить, необходимо ввести уникальное имя. После этого идёт возврат к шагу 1.**

Во всех таблицах набор операций одинаков: поиск и вставка.

**Неупорядоченная таблица:**

**Вставка:** добавление элемента в конец вектора.

**Поиск:** линейный поиск в векторе (прохождение по всем элементам).

**Упорядоченная таблица:**

**Вставка:** поиск нужного места в дереве и добавление новой записи в качестве новой записи в качестве потомка.

**Поиск:** начиная с корня, перемещаемся в левое поддерево, если искомый ключ меньше текущего, в правое – если больше текущего. Если значение совпадает, запись найдена.

**Хеш-таблица:**

**Вставка:** вычисляем хеш от ключа, проверяем, есть ли в списке запись с данным ключом. Если есть, заменяем значение, если нет – добавляем в конец списка новую запись.

**Поиск:** вычисляем хеш от ключа и ищем запись в списке.

**5. Эксперименты.**

В первом эксперименте мы сами вводим многочлен и выбираем операцию «Сложение». Далее вводим второй многочлен. На экран выводится результат операции, а также появляется выбор для пользователя: сохранить в таблицу результат или нет. Далее вводим уникальное имя для результата суммы многочленов. Затем программа выводит на экран сообщение о том, что данные успешно добавлены во все виды таблиц.

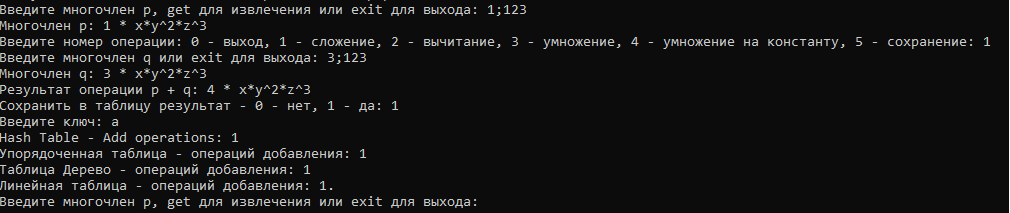
****

Рис.6 Консоль пользователя.

Во втором эксперименте мы вводим два многочлены и выбираем операцию «Вычитание». На экран выводится результат операции. Сохраняем данные в таблицы. Вводим уникальное имя для результата вычитания. Программа выводит сообщение, что данные успешно добавлены в таблицы. Затем мы пишем в консоли слово «get» и вводим уникальное имя. На экран выводится сообщение о том, что поиск и вывод в консоль желаемого многочлена выполнен успешно.

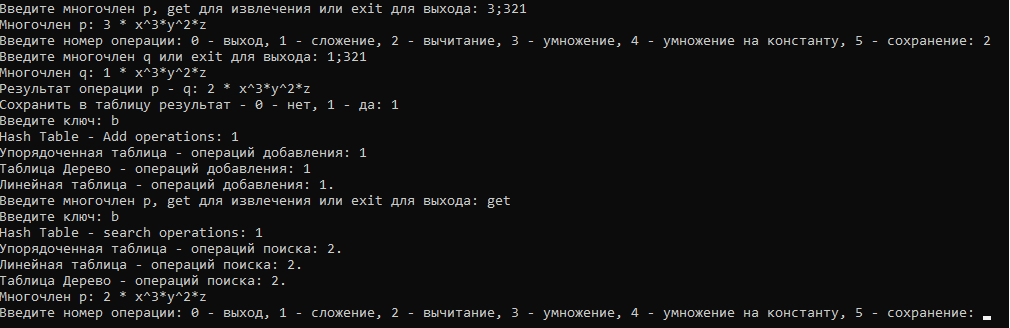


Рис.7 Консоль пользователя.

В третьем эксперименте мы вводим многочлен и выбираем операцию «Умножение на константу». Далее вводим второй многочлен. На экран выводится результат операции, а также появляется выбор для пользователя: сохранить в таблицу результат или нет. Далее вводим уникальное имя для результата. Затем программа выводит на экран сообщение о том, что данные успешно добавлены во все виды таблиц.

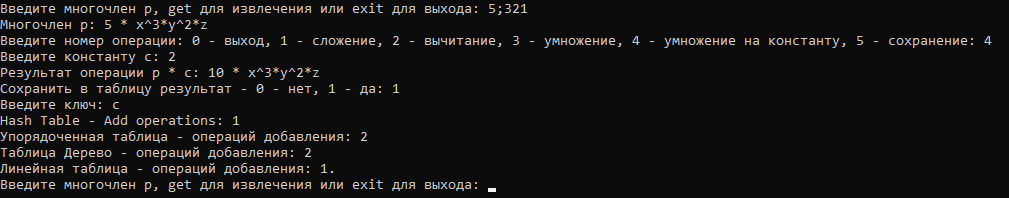


Рис.8 Консоль пользователя.

В четвёртом эксперименте мы вводим многочлен и вводим номер операции, которой не существует. Программа просит ввести значение от 0 до 5.

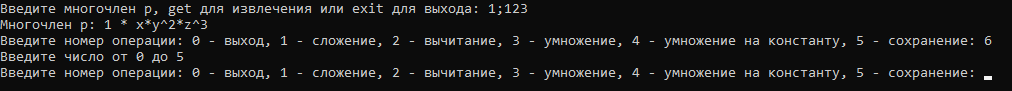


Рис.9 Консоль пользователя

В пятом эксперименте мы пытаемся ввести многочлен со степенями, превышающими 9. Программа выводит сообщение «Неверный формат ввода!».



Рис.10 Консоль пользователя

Также в качестве эксперимента были добавлены Google Tests для проверки работоспособности основных методов для каждой из таблиц, а также работоспособность основной функции main.

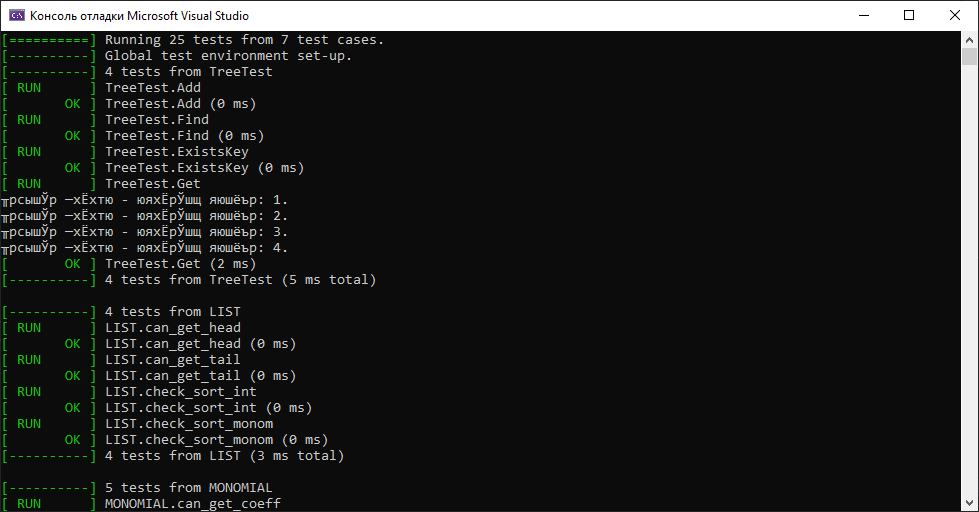


Рис.11 Консоль вывода тестов(1).

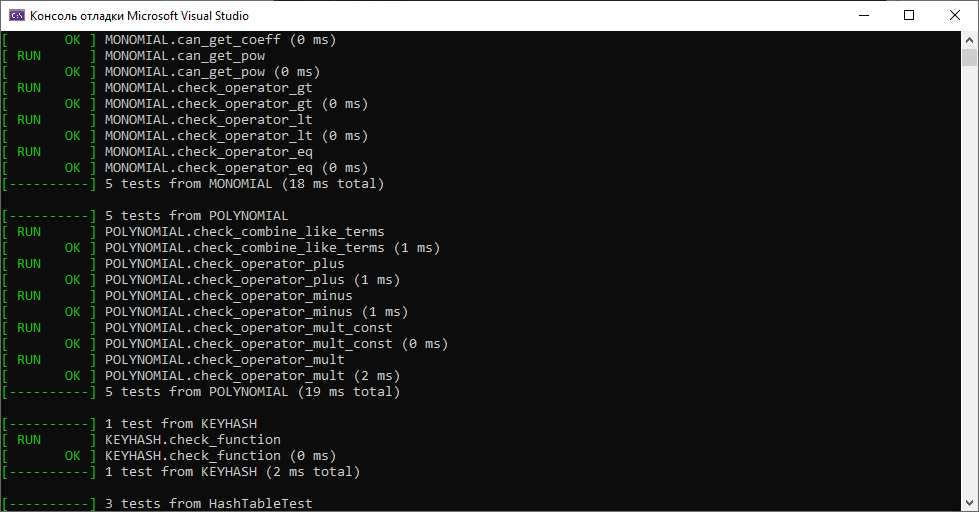
****

Рис.12 Консоль вывода тестов(2).

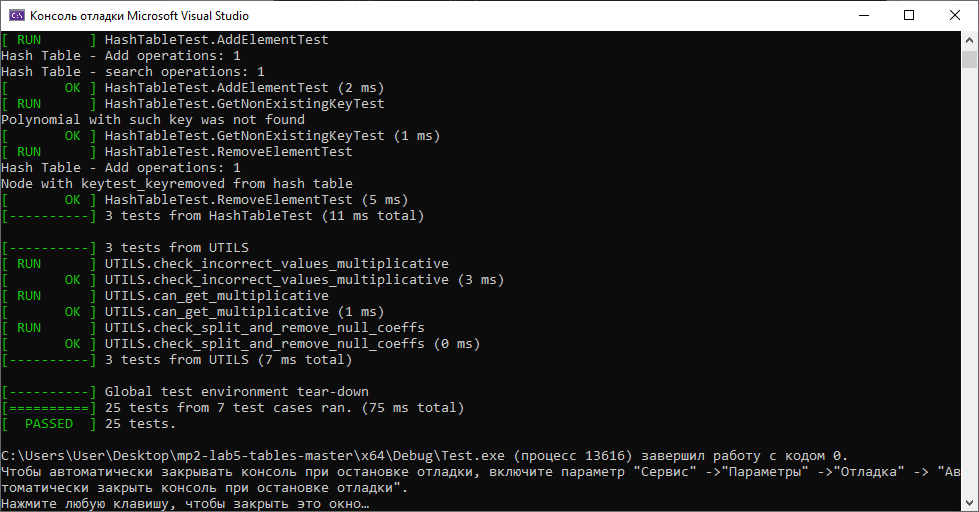


Рис.13 Консоль вывода тестов(3).

**6. Заключение.**

В лабораторной работе были разработаны классы неупорядоченных, упорядоченных, хеш-таблиц и дерева, а также вспомогательные классы для них. Их корректность проверена на тестах, созданных при помощи Google Tests. Также разработано пользовательское приложение, позволяющее совершать арифметические операции над многочленами и сохранять их с именами в таблице.

**7. Литература.**

1) Т. А. Павловская. C/C++ Программирование на языке высокого уровня. Режим доступа: <http://cph.phys.spbu.ru/documents/First/books/7.pdf>

2) Бьерн Страуструп. Язык программирования С++. Режим доступа: <http://8361.ru/6sem/books/Straustrup-Yazyk_programmirovaniya_c.pdf>

**8. Приложения.**

***8.1 Приложение 1.***

#include "polynomial.h"

#include "hashtable.h"

#include "orderedtable.h"

#include "LineTable.h"

#include "TreeTable.h"

#include <iostream>

#include <string>

int main()

{

setlocale(LC\_CTYPE, "Russian");

std::cout << "Пример:\n";

std::cout << "для полинома 4 \* x ^ 4 \* y ^ 2 \* z ввести нужно 4;421\n";

std::cout << "для полинома x\*y\*z + y\*z ^ 5 ввести нужно 1;111;1;15\n";

std::cout << "То есть для каждого монома первое число - коэффициент при одночлене, \nзатем точка с запятой, после число от 0 до 999 - показатели\n";

std::cout << "105 = x ^ 1 y ^ 0 z ^ 5\n";

std::cout << "256 = x ^ 2 y ^ 5 z ^ 6\n";

std::cout << "8 = z ^ 8\n";

std::cout << "60 = y ^ 6\n";

std::cout << "200 = x ^ 2\n";

std::cout << "Вводить полином нужно без пробелов\nВ случае, если полином введен неправильно-он приравнивается к 0 или выводится сообщение об ошибке\n";

HashTable<std::string, Polynomial, KeyHash> \*ht

= new HashTable<std::string, Polynomial, KeyHash>();

OrderedTable \*ot = new OrderedTable();

LineTable \*ut = new LineTable();

Tree\* tt = new Tree();

while (true)

{

try

{

std::cout << "Введите многочлен p, get для извлечения или exit для выхода: ";

std::string s1;

std::cin >> s1;

Polynomial p;

if (s1 == "exit")

break;

else if (s1 == "get")

{

std::string key;

std::cout << "Введите ключ: ";

std::cin >> key;

if (!ht->get(key, p))

continue;

ot->get(key);

ut->get(key);

tt->get(key);

}

else

p = Polynomial(s1);

std::cout << "Многочлен p: " << p << std::endl;

while (true)

{

bool b = true;

try

{

std::string s;

std::cout << "Введите номер операции: 0 - выход, 1 - сложение, 2 - вычитание, "

<< "3 - умножение, 4 - умножение на константу, 5 - сохранение: ";

std::cin >> s;

int op = atoi(s.c\_str());

if (op < 0 || op > 4)

throw std::exception();

else if (op == 0)

break;

else

{

if (op < 4)

{

while (true)

{

try

{

std::cout << "Введите многочлен q или exit для выхода: ";

std::string s2;

std::cin >> s2;

if (s2 == "exit")

break;

Polynomial q = Polynomial(s2), r;

std::cout << "Многочлен q: " << q << std::endl;

std::cout << "Результат операции ";

if (op == 1)

{

std::cout << "p + q: ";

r = p + q;

}

else if (op == 2)

{

std::cout << "p - q: ";

r = p - q;

}

else

{

std::cout << "p \* q: ";

r = p \* q;

}

std::cout << r << std::endl;

b = false;

int save;

std::cout << "Сохранить в таблицу результат - 0 - нет, 1 - да: ";

std::cin >> save;

if (save == 1)

{

std::string key;

std::cout << "Введите ключ: ";

std::cin >> key;

TableElement \*te = new TableElement(key, r);

if (!ht->existsKey(key))

ht->add(te->key(), te->value());

else

std::cout << "Такой ключ в хеш-таблице уже существует!" << std::endl;

if (!ot->existsKey(\*te))

ot->add(\*te);

else

std::cout << "Такой ключ в упорядоченной таблице уже существует!" << std::endl;

if (!ut->existsKey(key))

ut->add(\*te);

else

std::cout << "Такой ключ в линейной таблице уже существует!" << std::endl;

if (!tt->existsKey(\*te))

tt->add(\*te);

else

std::cout << "Такой ключ в дереве уже существует!" << std::endl;

}

break;

}

catch (std::overflow\_error& e)

{

std::cout << e.what() << std::endl;

}

catch (std::exception&)

{

std::cout << "Неверный формат ввода! Повторите ввод." << std::endl;

}

}

if (!b)

break;

}

else if (op == 4)

{

while (true)

{

try

{

std::string s;

std::cout << "Введите константу с: ";

std::cin >> s;

double c = atof(s.c\_str());

Polynomial r = p \* c;

std::cout << "Результат операции p \* c: " << r << std::endl;

b = false;

int save;

std::cout << "Сохранить в таблицу результат - 0 - нет, 1 - да: ";

std::cin >> save;

if (save == 1)

{

std::string key;

std::cout << "Введите ключ: ";

std::cin >> key;

TableElement \*te = new TableElement(key, r);

ht->add(te->key(), te->value());

ot->add(\*te);

ut->add(\*te);

tt->add(\*te);

}

break;

}

catch (std::exception&)

{

std::cout << "Неверный формат ввода! Повторите ввод." << std::endl;

}

}

if (!b)

break;

}

else

{

std::string key;

std::cout << "Введите ключ: ";

std::cin >> key;

TableElement \*te = new TableElement(key, p);

if (!ht->existsKey(key))

ht->add(te->key(), te->value());

else

std::cout << "Такой ключ в хеш-таблице уже существует!" << std::endl;

if (!ot->existsKey(\*te))

ot->add(\*te);

else

std::cout << "Такой ключ в упорядоченной таблице уже существует!" << std::endl;

if (!ut->existsKey(key))

ut->add(\*te);

else

std::cout << "Такой ключ в линейной таблице уже существует!" << std::endl;

if (!tt->existsKey(\*te))

tt->add(\*te);

else

std::cout << "Такой ключ в дереве уже существует!" << std::endl;

}

break;

}

}

catch (std::exception&)

{

std::cout << "Введите число от 0 до 5" << std::endl;

std::cin.clear();

}

}

}

catch (std::exception&)

{

std::cout << "Неверный формат ввода! Повторите ввод." << std::endl;

}

}

delete ht;

delete ot;

delete ut;

delete tt;

}

***8.2 Приложение 2***

#ifndef LIST\_H

#define LIST\_H

/\*

Класс "двусвязный список"

\*/

#include "Node.h"

template<typename T>

class List

{

Node<T> \*head, \*tail; // Начальный и конечный элементы

public:

// Конструктор класса

List()

{

head = tail = nullptr;

}

// Получение начального элемента

Node<T>\* getHead()

{

return head;

}

// Получение конечного элемента

Node<T>\* getTail()

{

return tail;

}

// Добавление элемента item в список

void push\_back(const T& item)

{

Node<T> \*tmp = new Node<T>;

tmp->next = nullptr;

tmp->data = item;

if (head != nullptr)

{

tmp->prev = tail;

tail->next = tmp;

tail = tmp;

}

else

{

tmp->prev = nullptr;

head = tail = tmp;

}

}

void clear()

{

head = tail = nullptr;

}

// Сортировка списка

void sort()

{

Node<T> \*i = head;

if (i != nullptr)

while (i->next != nullptr)

{

Node<T> \*j = i->next;

while (j != nullptr)

{

if (i->data < j->data)

{

T tmp = i->data;

i->data = j->data;

j->data = tmp;

}

j = j->next;

}

i = i->next;

}

}

};

#endif /\* LIST\_H \*/***.***

***8.3 Приложение 3.***

#ifndef MONOMIAL\_H

#define MONOMIAL\_H

#include <string>

// Класс "одночлен"

class Monomial

{

double coeff; // Коэффициент

int pow; // Показатели x, y, z

public:

// Конструктор по умолчанию

Monomial();

// Конструктор с параметрами

Monomial(double coeff, int pow);

// Получение коэффициента

double getCoeff() const;

// Получение степени

int getPow() const;

// Изменение коэффициента

void setCoeff(double coeff);

// Изменение степени

void setPow(int pow);

// Перегрузка операторов сравнения <, >, =

friend bool operator<(const Monomial& m1, const Monomial& m2);

friend bool operator>(const Monomial& m1, const Monomial& m2);

friend bool operator==(const Monomial& m1, const Monomial& m2);

// Перегрузка оператора потокового вывода

friend std::ostream& operator<<(std::ostream& os, const Monomial& m);

};

#endif /\* MONOMIAL\_H \*/

***8.4 Приложение 4.***

#ifndef POLYNOMIAL\_H

#define POLYNOMIAL\_H

#include "list.h"

#include "monomial.h"

#include <string>

// Класс "многочлен"

class Polynomial

{

// Список одночленов

List<Monomial>\* monomials;

// Упорядочивание многочлена

void arrange();

// Приведение подобных слагаемых

void combineLikeTerms();

public:

// Конструктор по умолчанию

Polynomial();

// Конструктор (параметр - список мономов)

Polynomial(const List<Monomial>& monomials);

// Конструктор (параметр - строка)

Polynomial(std::string s);

// Получение списка мономов

List<Monomial>\* getMonomials() const;

// Добавление одночлена

void addMonomial(const Monomial& m);

// Перегрузка арифметических операторов

friend Polynomial operator+(const Polynomial& p1, const Polynomial& p2);

friend Polynomial operator-(const Polynomial& p1, const Polynomial& p2);

friend Polynomial operator\*(const Polynomial& p1, const Polynomial& p2);

friend Polynomial operator\*(const Polynomial& p, double c);

friend Polynomial operator\*(double c, const Polynomial& p);

// Перегрузка оператора потокового вывода

friend std::ostream& operator<<(std::ostream& os, const Polynomial& p);

// Текстовое представление многочлена

std::string to\_string();

};

#endif /\* POLYNOMIAL\_H \*/

***8.5 Приложение 5.***

#ifndef HASHTABLE\_H

#define HASHTABLE\_H

#include "hashNode.h"

#include <sstream>

// Константа размера хэш-таблицы

const size\_t SIZE = 100;

// Хэш-функция

struct KeyHash

{

unsigned long operator()(const std::string& key) const

{

int sum = 0;

for (char c : key)

sum += (int)c;

return sum % SIZE;

}

};

// Класс хэш-таблицы

template <typename K, typename V, typename F>

class HashTable

{

HashNode<K, V> \*\*table; // Таблица

F function; // Хэш-функция

public:

// Конструктор хэш-таблицы

HashTable()

{

table = new HashNode<K, V>\*[SIZE]();

}

bool existsKey(const K &key)

{

if (table[function(key)])

return true;

else

return false;

}

// Поиск по ключу key и запись в value

bool get(const K &key, V &value)

{

// Вычисление хэш-значения

unsigned long hashValue = function(key);

HashNode<K, V> \*entry = table[hashValue];

int count = 0;

while (entry)

{

count++;

if (entry->key() == key) // Если найдена запись

{

value = entry->value();

std::cout << "Hash Table - search operations: " << count << std::endl;

return true;

}

entry = entry->next();

}

std::cout << "Polynomial with such key was not found\n";

return false;

}

void add(const K &key, const V &value)

{

// Вычисление хэш-значения

unsigned long hashValue = function(key);

HashNode<K, V> \*prev = NULL;

HashNode<K, V> \*entry = table[hashValue];

// Вычисление количества операций

int count = 1;

while (entry && entry->key() != key)

{

prev = entry;

entry = entry->next();

count++;

}

// Изменение таблицы (перезапись или создание узла)

if (!entry)

{

entry = new HashNode<K, V>(key, value);

if (prev == NULL)

table[hashValue] = entry;

else

prev->setNext(entry);

}

else

entry->setValue(value);

std::cout << "Hash Table - Add operations: "

<< count << std::endl;

}

//Метод удаления по ключу

void remove(const K& key)

{

// Вычисление хэш-значения

unsigned long hashValue = function(key);

HashNode<K, V>\* prev = nullptr;

HashNode<K, V>\* entry = table[hashValue];

// Поиск узла с указанным ключом

while (entry && entry->key() != key)

{

prev = entry;

entry = entry->next();

}

if (!entry)

{

// Узел с указанным ключом не найден

std::cout << "Node with key" << key << " not found." << std::endl;

return;

}

// Если узел найден, удаляем его из таблицы

if (prev == nullptr)

{

// Узел находится в начале списка

table[hashValue] = entry->next();

}

else

{

// Узел находится в середине или конце списка

prev->setNext(entry->next());

}

delete entry;

std::cout << "Node with key" << key << "removed from hash table" << std::endl;

}

};

#endif /\* HASHTABLE\_H \*/

***8.6 Приложение 6.***

#ifndef ORDERED\_TABLE\_H

#define ORDERED\_TABLE\_H

#include "../include/tableelement.h"

#include "TreeTable.h"

#include <iostream>

// Класс "упорядоченная таблица"

class OrderedTable

{

Tree \*table; // Бинарное дерево поиска

public:

// Конструктор таблицы

OrderedTable()

{

table = new Tree();

}

bool existsKey(const TableElement& te)

{

return table->existsKey(te);

}

// Добавление данных

void add(const std::string& key, const Polynomial& p)

{

int count = table->add(TableElement(key, p));

std::cout << "Упорядоченная таблица - операций добавления: "

<< count << std::endl;

std::cout << "Таблица Дерево - операций добавления: "

<< count << std::endl;

}

// Добавление данных

void add(const TableElement& te)

{

add(te.key(), te.value());

}

// Поиск полинома

Polynomial get(const std::string& key)

{

auto result = table->find(TableElement(key, Polynomial()));

std::cout << "Упорядоченная таблица - операций поиска: "

<< result.second << "." << std::endl;

return result.first->data.value();

}

};

#endif /\* ORDERED\_TABLE\_H \*/

***8.7 Приложение 7.***

#ifndef UNORDERED\_TABLE\_H

#define UNORDERED\_TABLE\_H

#include <iostream>

#include <vector>

#include "../include/tableelement.h"

// Класс "Линейная таблица"

class LineTable

{

std::vector<TableElement\*> table; // Массив

public:

// Конструктор таблицы

LineTable()

{

table = std::vector<TableElement\*>();

}

bool existsKey(const std::string& key)

{

for (auto te : table)

if (te->key() == key)

return true;

return false;

}

// Добавление данных

void add(const std::string& key, const Polynomial& p)

{

table.push\_back(new TableElement(key, p));

std::cout << "Линейная таблица - операций добавления: 1." << std::endl;

}

// Добавление данных

void add(const TableElement& te)

{

add(te.key(), te.value());

}

// Получение данных

Polynomial get(const std::string& key)

{

for (size\_t i = 0; i < table.size(); i++)

{

if (table.at(i)->key() == key)

{

std::cout << "Линейная таблица - операций поиска: "

<< i + 1 << "." << std::endl;

return table.at(i)->value();

}

}

return NULL;

}

};

***8.8 Приложение 8.***

#ifndef Tree\_H

#define Tree\_H

#include <utility>

#include <sstream>

#include <iostream>

#include "TreeNode.h"

#include "tableelement.h"

using namespace std;

// Класс "дерево"

class Tree

{

TreeNode<TableElement>\* root; // Корень

Tree\* table;

// Добавление в дерево

int \_add(TreeNode<TableElement>\* current, const TableElement& data, int count)

{

// Если пуст, то создание нового узла

if (current == NULL)

{

current = new TreeNode<TableElement>(data);

return count + 1;

}

else if (current->data < data)

{

if (current->right == NULL)

{

current->right = new TreeNode<TableElement>(data);

return count + 1;

}

else

return \_add(current->right, data, count + 1);

}

else

{

if (current->left == NULL)

{

current->left = new TreeNode<TableElement>(data);

return count + 1;

}

else

return \_add(current->left, data, count + 1);

}

}

// Поиск по дереву

pair<TreeNode<TableElement>\*, int> \_find(TreeNode<TableElement>\* current,

const TableElement& key, int count)

{

// Если текущий пуст или нашли ключ, то возвращаем текущий

if (current == NULL || current->data == key)

return pair<TreeNode<TableElement>\*, int>(current, count + 1);

// Иначе переход в левое или правое поддерево

else if (current->data < key)

return \_find(current->right, key, count + 1);

else

return \_find(current->left, key, count + 1);

}

bool \_existsKey(TreeNode<TableElement>\* current, const TableElement& te)

{

if (current == NULL)

return false;

else if (current->data.key() == te.key())

return true;

else if (current->data < te)

return \_existsKey(current->right, te);

else

return \_existsKey(current->left, te);

}

public:

// Конструктор по умолчанию

Tree()

{

root = nullptr;

table = nullptr;

}

// Конструктор с параметром

Tree(const TableElement& data)

{

root = new TreeNode<TableElement>(data);

table = new Tree;

}

// Добавление данных

auto add(const TableElement& data)

{

// Если корень пуст, создаем его

if (root == NULL)

{

root = new TreeNode<TableElement>(data);

return 1;

}

else

return \_add(root, data, 0);

}

// Поиск по ключу

pair<TreeNode<TableElement>\*, int> find(const TableElement& key)

{

return \_find(root, key, 0);

}

bool existsKey(const TableElement& te)

{

return \_existsKey(root, te);

}

// Получение значения по ключу

TableElement get(const std::string& key)

{

// Ищем элемент по ключу

auto result = find(TableElement(key, Polynomial()));

cout << "Таблица Дерево - операций поиска: "

<< result.second << "." << std::endl;

// Если элемент найден, возвращаем его значение

if (result.first != nullptr)

return result.first->data;

// Иначе возвращаем пустой элемент

else

return TableElement();

}

void add(const std::string& key, const Polynomial& p)

{

int count = table->add(TableElement(key, p));

cout << "Таблица Дерево - операций добавления: 1 "

<< count << std::endl;

}

};

#endif // !Tree\_H