Министерство науки и высшего образования РФ

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

Институт информационных технологий, математики и механики

Отчет по учебной практике

**Аналитические преобразования полиномов  
от нескольких переменных**

Выполнил: Параничева Алена Владиславовна

студент группы 381806-1

Проверил: к. т. н., доцент кафедры МОСТ

Кустикова Валентина Дмитриевна

Нижний Новгород

2020

Оглавление

[2. Введение 3](#_Toc37094422)

[3. Постановка задачи 4](#_Toc37094423)

[4. Руководство пользователя 5](#_Toc37094424)

[5. Руководство программиста 6](#_Toc37094425)

[5.1. Структура программы 6](#_Toc37094426)

[5.2. Описание структур данных 6](#_Toc37094427)

[5.2.1. Класс TNode 6](#_Toc37094428)

[5.2.2. Класс TList 7](#_Toc37094429)

[5.2.3. Класс TNode<double, unsigned int> 10](#_Toc37094430)

[5.2.4. Класс TList<double, unsigned int> 14](#_Toc37094431)

[5.2.5. Класс Polinom 17](#_Toc37094432)

[5.3. Описание алгоритмов 21](#_Toc37094433)

[5.3.1. Представление связного списка 21](#_Toc37094434)

[5.3.2. Представление структуры полином. 23](#_Toc37094435)

[6. Заключение 25](#_Toc37094436)

[7. Литература 26](#_Toc37094437)

[8. Приложение 27](#_Toc37094438)

# Введение

Если обычный человек задумается о применимости полиномов в жизни, то ему может показаться, что они весьма бесполезны, однако это не так. С помощью них можно решать не только задачи дискретной математики, квантовой механики или математического анализа, но и, например, использовать полиномы как способ передачи данных.

Полином – это сумма мономов, где моном – это частный случай многочлена.

Для того, чтобы реализовать программу с преобразованиями полиномов, можно использовать такую структуру данных как связанный список. Связный список — базовая динамическая структура данных, состоящая из узлов (в нашей программе мономов), каждый из которых содержит как собственно данные, так и одну или две ссылки («связки») на следующий и/или предыдущий узел списка.

Поэтому в программе будет представлены классы Узла(Монома), Листа и Полинома.

# Постановка задачи

Постановка задачи:

Разработать программу, позволяющую проводить преобразования над полиномами, где вид каждого монома можно представить так: С\*xn\*yk\*zs.

Программа должна уметь: обрабатывать вводимый полином, упрощать, складывать, вычитать, умножать полиномы, выводить результат и другие простейшие операции.

Входные данные:

* Строка с полиномом

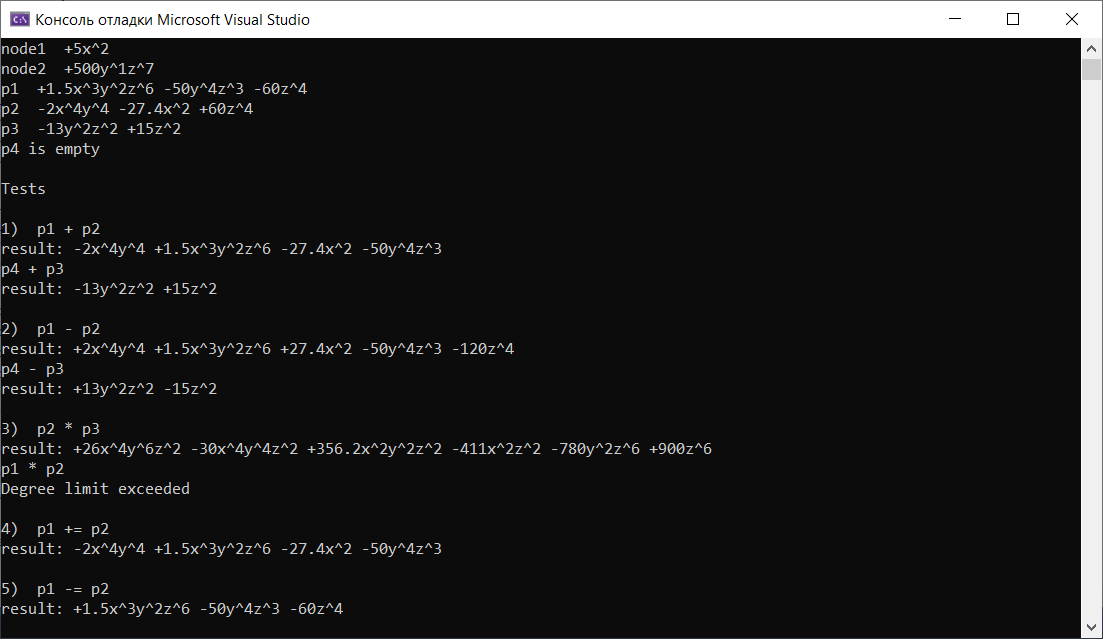
Выход:

* Итог всех преобразований над полиномом

# Руководство пользователя

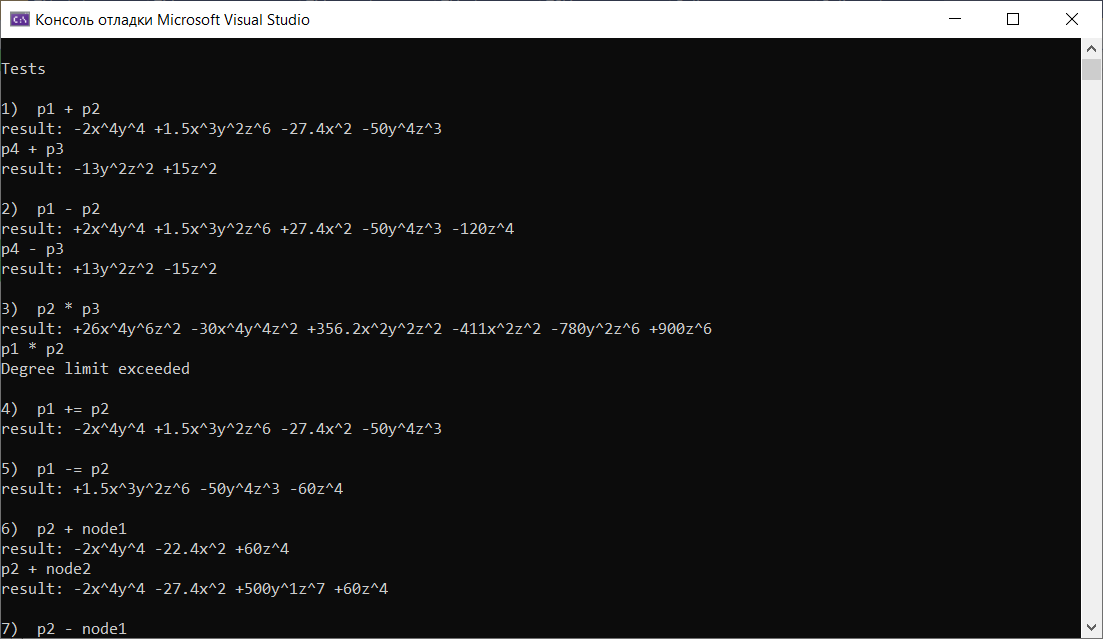
В данном руководстве содержатся пошаговые инструкции по работе с программой:

Запустите программу. Перед вами отобразится экран, на котором выведены переменные, которые будут участвовать в операциях над полиномами (рис. 1).



1. Программа после запуска.

Далее вы увидите операции над полиномами и их результаты. Возникающие исключительные ситуации также появятся на экране (рис. 2).



1. Сообщение об ошибке при вводе выражения с некорректной записью.

# Руководство программиста

## Структура программы

Код программы находится в файлах:

**Main.cpp** содержит основную функцию с тестом программы,

**TNode.h** – заголовочный файл, содержащий реализацию узла списка и связного списка,

**Monom.h** – заголовочный файл, содержащий специализацию класса узла для работы с мономами,

**TSList.h** - заголовочный файл, содержащий специализацию класса списка для работы с полиномами

**Polinom.h** – заголовочный файл, содержащий реализацию класса полином

## Описание структур данных

### Класс TNode

template<typename TData, typename TKey>

class TNode

{

public:

TData\* pData;

TKey key;

TNode\* pNext;

TNode();

TNode(TKey, TData\*, TNode\* \_pNext = nullptr);

TNode(const TNode&);

friend std::ostream& operator << (std::ostream&, TNode<TData, TKey>&);

};

*Поля класса:*

TData\* pData – указатель на область памяти, где хранятся данные.

TKey key – переменная ключ.

TNode< TData, TKey>\* pNext – указатель на следующий узел.

*Методы класса:*

TNode()

***Назначение:*** конструктор по умолчанию.

***Входные параметры***: отсутствуют.

TNode(TKey, TData\*, TNode\* \_pNext = nullptr)

***Назначение:*** конструктор с параметрами (ключ, данные, указатель на следующий узел).

***Входные параметры:*** параметры типа TKey, TData\*, TNode\*.

TNode(const TNode&)

***Назначение:*** конструктор копирования.

***Входные параметры:*** параметр типа const TNode&.

friend std::ostream& operator<<(std::ostream&, TNode<TData, TKey>&)

***Назначение:*** перегрузка операции вывода.

***Входные параметры:*** параметры типа std::ostream&, TNode<TData, TKey>&.

***Выходные параметры:*** параметр типа std::ostream&.

### Класс TList

template<typename TData, typename TKey>

class TList

{

private:

TNode<typename TData, typename TKey>\* pFirst;

TNode<typename TData, typename TKey>\* pPrev;

TNode<typename TData, typename TKey>\* pCur;

TNode<typename TData, typename TKey>\* pNext;

public:

TList();

TList(const TNode<TData, TKey>\*);

TList(const TList&);

~TList();

bool IsEnded() const;

void Next();

void Reset();

TNode<TData, TKey>\* Search(TKey);

void InsertEnd(TKey, TData\*);

void InsertFirst(TKey, TData\*);

void InsertAfter(TKey, TData\*, TKey);

void InsertBefore(TKey, TData\*, TKey);

void Remove(TKey);

friend std::ostream& operator << (std::ostream&, TList<TData, TKey>&);

};

*Поля класса:*

TNode<typename TData, typename TKey>\* pFirst – указатель на первый элемент списка.

TNode<typename TData, typename TKey>\* pPrev – указатель на предыдущий элемент списка.

TNode<typename TData, typename TKey>\* pCur – указатель на текущий элемент списка.

TNode<typename TData, typename TKey>\* pNext – указатель на следующий элемент списка.

*Методы класса:*

TList()

***Назначение:*** конструктор по умолчанию.

***Входные параметры:*** отсутствуют.

TList(const TNode<TData, TKey>\*)

***Назначение:*** конструктор с параметрами.

***Входные параметры:*** параметры типа const TNode<TData, TKey>\*(указатель на первый узел).

TList(const TList&)

***Назначение:*** конструктор копирования.

***Входные параметры:*** параметр типа const TList&.

~TList()

***Назначение:*** удаление объекта класса TList.

bool IsEnded() const

***Назначение:*** метод, проверяющий достигнут ли конец списка.

Входные параметры: отсутствуют.

***Выходные параметры:*** True – дошли до конца до конца списка, False – иначе.

void Next()

***Назначение:*** метод, переставляющий указатель на текущее звено на следующее за ним.

***Входные параметры:*** отсутствуют.

***Выходные параметры:*** отсутствуют.

void Reset()

***Назначение:*** метод, который возвращает указатель на текущее звено в начало.

***Входные параметры:*** отсутствуют.

***Выходные параметры:*** отсутствуют.

TNode<TData, TKey>\* Search(TKey)

***Назначение:*** метод, поиска звена по заданному ключу.

***Входные параметры:*** параметр типа TKey(ключ звена, который необходимо найти.

***Выходные параметры:*** найденный элемент.

void InsertEnd(TKey, TData\*)

***Назначение:***  метод вставки нового звена в конец списка. Параметры: ключ нового звена, данные нового звена.

***Входные параметры:*** параметры типа TKey, TData\*(ключ нового звена, данные нового звена).

***Выходные параметры:*** отсутствуют.

void InsertFirst (TKey, TData\*)

***Назначение:*** метод вставки нового звена в начало списка.

***Входные параметры:*** параметры типа TKey, TData\* (ключ нового звена, данные нового звена).

***Выходные параметры:*** отсутствуют.

void InsertAfter(TKey, TKey, TData\*)

***Назначение:*** метод вставки звена после звена с заданным ключом.

***Входные параметры:*** параметры типа TKey, TKey, TData\* (ключ искомого звена, ключ нового звена, данные нового звена).

Выходные параметры: отсутствуют.

void InsertBefore(TKey, TKey, TData\*)

***Назначение:*** метод вставки звена перед звеном с заданным ключом.

***Входные параметры:*** параметры типа TKey, TKey, TData\* (ключ искомого звена, ключ нового звена, данные нового звена).

Выходные параметры: отсутствуют.

void Remove(TKey)

***Назначение:*** метод удаления звена с заданным ключом.

***Входные параметры:*** Параметр типа TKey (ключ удаляемого звена).

Выходные параметры: отсутствуют.

friend std::ostream& operator<<(std::ostream&, TList<TData, TKey>&)

***Назначение:*** перегрузка операции вывода.

***Входные параметры:*** параметры типа std::ostream&, TList<TData, TKey>&.

***Выходные параметры:*** параметр типа std::ostream&.

### Класс TNode<double, unsigned int>

template<>

class TNode<double, unsigned int>

{

public:

double pData;

unsigned int key;

TNode<double, unsigned int>\* pNext;

TNode();

TNode(unsigned int, double, TNode\* \_pNext = nullptr);

TNode(const TNode&);

TNode& operator=(const TNode&);

TNode operator+(const TNode&);

TNode operator-(const TNode&);

TNode operator\*(const TNode&);

TNode& operator+=(const TNode&);

TNode& operator-=(const TNode&);

TNode& operator\*=(const TNode&);

TNode& operator\*=(double);

TNode operator\*(double);

TNode operator-() const;

bool operator==(const TNode&) const;

bool operator!=(const TNode&) const;

friend ostream& operator<<(ostream&, TNode&);

};

*Поля класса:*

unsigned int key – переменная, ключ.

double pData – указатель на область памяти, где хранятся данные.

TNode<double, unsigned int>\* pNext – указатель на следующий узел.

*Методы класса:*

TNode()

***Назначение:*** конструктор по умолчанию.

***Входные параметры:*** отсутствуют.

TNode(unsigned int, double, TNode\* \_pNext = nullptr)

***Назначение:*** конструктор с параметрами (ключ, данные, указатель на следующий узел).

***Входные параметры:*** параметры типа unsigned int, double, TNode\* \_pNext = nullptr.

TNode(const TNode&)

***Назначение:*** конструктор копирования.

***Входные параметры:*** параметр типа const TNode&.

~TNode()

***Назначение:*** удаление объекта класса TNode.

TNode& operator=(const TNode&)

***Назначение:*** перезрузка оператора присваивания.

***Входные параметры:*** параметр типа const TNode&.

***Выходные параметры:*** параметр типа TNode&.

TNode\* operator+(const TNode&)

***Назначение:*** перезрузка операции сложения.

***Входные параметры:*** параметр типа const TNode&.

***Выходные параметры:*** параметр типа TNode\*.

TNode\* operator-(const TNode&)

***Назначение:*** перезрузка операции вычитания.

***Входные параметры:*** параметр типа const TNode&.

***Выходные параметры:*** параметр типа TNode\*.

TNode\* operator\*(const TNode&)

***Назначение:*** перезрузка операции умножения.

***Входные параметры:*** параметр типа const TNode&.

***Выходные параметры:*** параметр типа TNode\*.

TNode\* operator+=(const TNode&)

***Назначение:*** перезрузка операции сложения, совмещенная с присваиванием.

***Входные параметры:*** параметр типа const TNode&.

***Выходные параметры:*** параметр типа TNode\*.

TNode\* operator-=(const TNode&)

***Назначение:*** перезрузка операции вычитания, совмещенная о с присваиванием.

***Входные параметры:*** параметр типа const TNode&.

***Выходные параметры:*** параметр типа TNode\*.

TNode\* operator\*=(const TNode&)

***Назначение:*** перезрузка операции умножения, совмещенная с присваиванием.

***Входные параметры:*** параметр типа const TNode&.

***Выходные параметры:*** параметр типа TNode\*.

TNode\* operator\*(double)

***Назначение:*** перезрузка операции умножения с константой.

***Входные параметры:*** параметр типа double.

***Выходные параметры:*** параметр типа TNode\*.

TNode\* operator\*=(double)

***Назначение:*** перезрузка операции умножения с константой, совмещенная с присваиванием.

***Входные параметры:*** параметр типа double.

***Выходные параметры:*** параметр типа TNode\*.

bool operator==(const TNode&) const

***Назначение:*** перегрузка сравнения.

***Входные параметры:*** параметр типа const TNode&.

***Выходные параметры:*** True – узлы равны, False – иначе.

bool operator!=(const TNode&) const

***Назначение:*** перегрузка операции «неравно».

***Входные параметры:*** параметр типа const TNode&.

***Выходные параметры:*** True – узлы не равны, False – иначе.

friend std::ostream& operator<<(std::ostream&, TNode<double, unsigned int>&)

***Назначение:*** перегрузка операции вывода.

***Входные параметры:*** параметры типа std::ostream&, TNode<double, unsigned int>&.

***Выходные параметры:*** параметр типа std::ostream&.

### Класс TList<double, unsigned int>

template<>

class TList<double, unsigned int>

{

private:

TNode<double, unsigned int>\* pFirst;

TNode<double, unsigned int>\* pPrev;

TNode<double, unsigned int>\* pCur;

TNode<double, unsigned int>\* pNext;

public:

TList();

TList(const TNode<double, unsigned int>\*);

TList(const TList&);

~TList();

bool operator==(const TList&) const;

bool IsEnded() const;

void Next();

void Reset();

TNode<double, unsigned int>\* Search(unsigned int);

void InsertEnd(unsigned int, double);

void InsertFirst(unsigned int, double);

void InsertAfter(unsigned int, double, unsigned int);

void InsertBefore(unsigned int, double, unsigned int);

void Remove(unsigned int);

friend std::ostream& operator << (std::ostream&, TList<double, unsigned int>&);

};

*Поля класса:*

TNode<double, unsigned int>\* pFirst – указатель на первый элемент списка.

TNode<double, unsigned int>\* pPrev – указатель на предыдущий элемент списка.

TNode<double, unsigned int>\* pCur – указатель на текущий элемент списка.

TNode<double, unsigned int>\* pNext – указатель на следующий элемент списка.

*Методы класса:*

TList()

***Назначение:*** конструктор по умолчанию.

***Входные параметры:*** отсутствуют.

TList(const TNode<double, unsigned int>\*)

***Назначение:*** конструктор с параметрами

***Входные параметры:*** параметры типа const TNode<TData, TKey>\*(указатель на первый узел).

TList(const TList&)

***Назначение:*** конструктор копирования.

***Входные параметры:*** параметр типа const TList&.

~TList()

***Назначение:*** удаление объекта класса TList.

bool operator==(const TList&) const

***Назначение:*** перегрузка сравнения.

***Входные параметры:*** параметр типа const TList&.

***Выходные параметры:*** True – листы равны, False – иначе.

bool IsEnded() const

***Назначение:*** метод, проверяющий достигнут ли конец списка.

***Входные параметры***: отсутствуют.

***Выходные параметры:*** True – дошли до конца до конца списка, False – иначе.

void Next()

***Назначение:*** метод, переставляющий указатель на текущее звено на следующее за ним.

***Входные параметры:*** отсутствуют.

***Выходные параметры:*** отсутствуют.

void Reset()

***Назначение:*** метод, который возвращает указатель на текущее звено в начало.

***Входные параметры:*** отсутствуют.

***Выходные параметры:*** отсутствуют.

TNode<double, unsigned int>\* Search(unsigned int)

***Назначение:*** метод, поиска звена по заданному ключу.

***Входные параметры:*** параметр типа unsigned int (ключ звена, который необходимо найти).

***Выходные параметры:*** найденное элемент.

void InsertEnd(unsigned int, double)

***Назначение:***  метод вставки нового звена в конец списка. Параметры – ключ нового звена, данные нового звена.

***Входные параметры:*** параметры типа unsigned int, double(ключ нового звена, данные нового звена).

***Выходные параметры:*** отсутствуют.

void InsertFirst(unsigned int, double)

***Назначение:*** метод вставки нового звена в начало списка.

***Входные параметры:*** параметры типа unsigned int, double (ключ нового звена, данные нового звена).

***Выходные параметры:*** отсутствуют.

void InsertAfter(unsigned int, double, unsigned int)

***Назначение:*** метод вставки звена после звена с заданным ключом.

***Входные параметры:*** параметры типа unsigned int, double, unsigned int (ключ искомого звена, ключ нового звена, данные нового звена).

***Выходные параметры:*** отсутствуют.

void InsertBefore(unsigned int, double, unsigned int)

***Назначение:*** метод вставки звена перед звеном с заданным ключом.

***Входные параметры:*** параметры типа unsigned int, double, unsigned int (ключ искомого звена, данные нового звена, ключ нового звена).

***Выходные параметры:*** отсутствуют.

void Remove(unsigned int)

***Назначение:*** метод удаления звена с заданным ключом.

***Входные параметры:*** Параметр типа unsigned int (ключ удаляемого звена).

***Выходные параметры:*** отсутствуют.

friend std::ostream& operator<<(std::ostream&, TList<double, unsigned int>&)

***Назначение:*** перегрузка операции вывода.

***Входные параметры:*** параметры типа std::ostream&, TList<double, unsigned int>&.

***Выходные параметры:*** параметр типа std::ostream&.

### Класс Polinom

class Polinom

{

private:

TList<double,unsigned int>\* pol;

public:

Polinom();

Polinom(const string&);

Polinom(TList<double, unsigned int>\*);

Polinom(const Polinom&);

~Polinom();

TList<double, unsigned int>\* StandartView();

bool operator==(const Polinom&) const;

Polinom& operator=(const Polinom&);

Polinom operator+(const Polinom&);

Polinom& operator+=(const Polinom&);

Polinom operator-(const Polinom&);

Polinom& operator-=(const Polinom&);

Polinom operator\*(const Polinom&);

Polinom operator+(const TNode<double, unsigned int>&);

Polinom operator-(const TNode<double, unsigned int>&);

Polinom operator\*(const TNode<double, unsigned int>&);

Polinom& operator+=(const TNode<double, unsigned int>&);

Polinom& operator-=(const TNode<double, unsigned int>&);

Polinom& operator\*=(const TNode<double, unsigned int>&);

Polinom operator-();

friend ostream& operator<<(ostream&, Polinom&);

};

*Поля класса:*

TList<double, unsigned int>\* pol - полином.

*Методы класса:*

Polinom()

***Назначение:*** конструктор по умолчанию.

***Входные параметры:*** отсутствуют.

Polinom(const string&)

***Назначение:*** конструктор с параметрами

***Входные параметры:*** параметр типа const string& (строка с полиномом).

Polinom(TList<double, unsigned int>\*);

***Назначение:*** конструктор с параметрами

***Входные параметры:*** параметр типа TList<double, unsigned int>\*).

Polinom(const Polinom&)

***Назначение:*** конструктор с параметрами

***Входные параметры:*** параметры типа const Polinom&.

~Polinom()

***Назначение:*** удаление объекта класса Polinom.

TList<double, unsigned int>\* StandartView()

***Назначение:*** метод преобразования полинома в стандарный вид (приведение мономов с одинаковыми степенями, сортировка полинома в порядке убывания степени монома).

***Входные параметры:*** отсутствуют.

***Выходные параметры:*** параметр типа TList<double, unsigned int>\*.

bool operator==(const Polinom&) const

***Назначение:*** перегрузка операции сравнения.

***Входные параметры:*** параметр типа const Polinom&.

***Выходные параметры:*** True – полиномы равны, False – иначе.

Polinom& operator=(const Polinom&)

***Назначение:*** перезрузка операции присваивания.

***Входные параметры:*** параметр типа const Polinom&.

***Выходные параметры:*** параметр типа Polinom&.

Polinom operator+(const Polinom&)

***Назначение:*** перезрузка операции сложения.

***Входные параметры:*** параметр типа const Polinom&.

***Выходные параметры:*** параметр типа Polinom.

Polinom& operator+=(const Polinom&)

***Назначение:*** перезрузка операции сложения, совмещенная с присваиванием.

***Входные параметры:*** параметр типа const Polinom&.

***Выходные параметры:*** параметр типа Polinom&.

Polinom operator-(const Polinom&)

***Назначение:*** перезрузка операции вычитания.

***Входные параметры:*** параметр типа const Polinom&.

***Выходные параметры:*** параметр типа Polinom.

Polinom& operator-=(const Polinom&)

***Назначение:*** перезрузка операции вычитания, совмещенная с присваиванием.

***Входные параметры:*** параметр типа const Polinom&.

***Выходные параметры:*** параметр типа Polinom&.

Polinom operator\*(const Polinom&)

***Назначение:*** перезрузка операции умножения.

***Входные параметры:*** параметр типа const Polinom&.

***Выходные параметры:*** параметр типа Polinom.

Polinom operator+(const TNode<double, unsigned int>&)

***Назначение:*** перезрузка операции сложения.

***Входные параметры:*** параметр типа const TNode<double, unsigned int>&.

***Выходные параметры:*** параметр типа Polinom.

Polinom operator-(const TNode<double, unsigned int>&)

***Назначение:*** перезрузка операции вычитания.

***Входные параметры:*** параметр типа const TNode<double, unsigned int>&.

***Выходные параметры:*** параметр типа Polinom.

Polinom operator\*(const TNode<double, unsigned int>&)

***Назначение:*** перезрузка операции умножения.

***Входные параметры:*** параметр типа const TNode<double, unsigned int>&.

***Выходные параметры:*** параметр типа Polinom.

Polinom& operator+=(const TNode<double, unsigned int>&)

***Назначение:*** перезрузка операции сложения, совмещенная с присваиванием.

***Входные параметры:*** параметр типа const TNode<double, unsigned int>&.

***Выходные параметры:*** параметр типа Polinom&.

Polinom& operator-=(const TNode<double, unsigned int>&)

***Назначение:*** перезрузка операции вычитания, совмещенная с присваиванием.

***Входные параметры:*** параметр типа const TNode<double, unsigned int>&.

***Выходные параметры:*** параметр типа Polinom&.

Polinom& operator\*=(const TNode<double, unsigned int>&)

***Назначение:*** перезрузка операции умножения, совмещенная с присваиванием.

***Входные параметры:*** параметр типа const TNode<double, unsigned int>&.

***Выходные параметры:*** параметр типа Polinom&.

Polinom operator-()

***Назначение:*** перезрузка операции «унарный минус» (меняюет знаки в полиноме на противоположные).

***Входные параметры:*** отсутствуют.

***Выходные параметры:*** параметр типа Polinom.

friend ostream& operator<<(ostream&, Polinom&)

***Назначение:*** перегрузка операции вывода.

***Входные параметры:*** параметры типа ostream&, Polinom&.

***Выходные параметры:*** параметр типа ostream&.

## Описание алгоритмов

### Представление связного списка

В программе используется такая структура данных, как связный список. Связный список — базовая динамическая структура данных, состоящая из узлов (в нашей программе мономов), каждый из которых содержит поле с данными, ключ и одну ссылку («связку») на следующий узел списка. Поле указателя последнего узла содержит нулевое значение.

Характеристиками связного списка являются:

1. Указатель на первый элемент списка
2. Указатель на предыдущий элемент списка
3. Указатель на текущий элемент списка
4. Указатель на следующий элемент списка

Операции над списком:

1. Проверка на нахождение последнего элемента списка.
   1. Возвращает true, если указатель на текущий элемент имеет нулевое значение, если это не так, возвращает false.
2. Переход к следующему элементу списка.
   1. Проверяем не находимся ли мы в конце списка. Если это так, бросаем исключение.
   2. Перебрасываем указатель текущего элемента на следующий элемент.
3. Возвращение к первому элементу списка.
   1. Присваиваем указателю на текущий элемент списка указатель на первый элемент списка.
4. Поиск элемента с заданным ключом.
   1. Выполняем обход по списку до тех пор, пока не встретится искомого звено.
   2. Если звено найдено – выходим из цикла и возвращаем найденное звено
   3. Если звено не найдено – возвращаем nullptr.
5. Вставка элемента в конец списка.
   1. Если список пустой, то указатель на начало списка ссылаем на новый элемент
   2. Если список не пустой, то доходим до конца списка
   3. указатель на следующее звено последнего звена ссылаем на новый элемент, у нового звена следующим делаем nullptr.
6. Вставка элемента в начало списка.
   1. Указатель на следующий элемент нового звена ссылаем на первое звено.
   2. Если список пустой, то ссылаем на nullptr.
   3. Указатель на начало списка ставим на новый элемент.
7. Вставка элемента после заданного элемента.
   1. Если список пустой, бросаем исключение.
   2. Находим в списке заданное звено.
   3. Его указатель на следующий ссылаем на новое звено.
   4. Указатель на следующий у нового звена ссылаем на следующий после заданного.
   5. Если заданное звено списка не было найдено, бросается исключение.
8. Вставка элемента перед заданным элементом.
   1. Если список пустой, бросаем исключение.
   2. Находим в списке заданное звено.
   3. Его предыдущему звену ссылаем указатель на следующий на новое звено
   4. Указатель на следующий нового звена ссылаем на заданное звено
   5. Если заданное звено списка не было найдено, бросается исключение.
9. Удаление элемента с заданным ключом
   1. Проверяем список на пустоту. Если он пуст, бросаем исключение.
   2. Ищем элемент, который нужно удалить. Если его нет в списке, бросаем исключение. Искомый элемент становится текущем.
   3. Если текущий совпадает с первым элементом, удаляем первый элемент. Указатель на первый элемент переходит к следующему.
   4. Если не равны, меняем указатель на следующий элемент у предыдущего элемента на следующий элемент текущего. Удаляем текущий.

### Представление структуры полином.

Структура данных полином основана на односвязном списке, где каждый узел списка является мономом. Ключ несет в себе значение – степени монома, где цифра в сотнях обозначает степень переменной Х, цифра в десятках степень Y, цифра в единицах – степень Z. При этом значение ключа не должно превышать 999. Данные являются значением коэффициента при мономе.

У полинома лишь одно поле – указатель на связный список.

Операции над полиномом:

1. Приведение полинома в стандартный вид: приведение мономов с одинаковыми степенями, сортировка полинома в порядке убывания степени монома (дополнительная операция).
2. Присваивание полиномов.
   1. Если указатель на текущий полином равен указателю на переданный, то возвращаем текущий полином.
   2. Если переданный полином не пустой, то удаляем все его мономы.
   3. Создаем новый список мономов с помощью конструктора копирования.
   4. Возвращаем текущий полином.
3. Сложение двух полиномов.
   1. Создаем результирующий полином.
   2. Сложение двух заданных полиномов будет проводится до тех пор, пока один из них не закончится.
   3. Если ключ текущего монома из первого полинома меньше, чем ключ текущего монома из второго полинома, то производится вставка в конец результирующего полинома звена из первого полинома.
   4. Иначе вставка в конец звена из второго полинома.
   5. Если ключи оказали равными, то смотрим чему равен коэффициент.
   6. Если это не ноль, то производится вставка в конец звена с новым коэффициентом.
   7. Далее передвигаем указатели текущего звена на следующие элементы.
   8. После того как цикл завершен, добавляем оставшиеся мономы не закончившегося полинома.
4. Вычитание двух полиномов.
   1. Сложение двух полиномов, с использованием над одним из них операцию унарного минуса.
5. Унарный минус.
   1. Все знаки коэффициентов у мономов меняются на противоположные.
6. Умножение полиномов.
   1. Реализовано с помощью перегрузки операции умножения у мономов.
   2. Проходим по одному полиному, последовательно перемножая текущий моном на мономы второго полинома.

# Заключение

В ходе практической работы была разработана и реализована программа, поддерживающую аналитические преобразования над полиномами от 3 переменных. В ходе работы были реализованы: шаблонный класс списка, шаблонный класс узла списка, их специализации, необходимые для структуры данных Полином. Также были успешно проведены тесты по работе с программой.

# Литература

1. *Т. Пратт, М. Зелковиц.* Языки программирования: разработка и реализация Terrence W. Pratt, Marvin V. Zelkowitz. Programming Languages: Design and Implementation. — 4-е издание. — Питер, 2002. — 688 с. — (Классика Computer Science). — 4000 экз.
2. Дмитрий Пискарев. Алгоритмы и структуры данных для начинающих: связный список. 21.10.2015 -[http://www.interface.ru/home.asp?artId=1492]
3. Егор Григорьев. Структура данных: стеки. 21.12.2013 - [http://cppstudio.com/post/5155/]
4. [Beau Carnes](https://medium.freecodecamp.org/10-common-data-structures-explained-with-videos-exercises-aaff6c06fb2b). 10 типов структур данных, которые нужно знать. 04.08.2017. –[<https://habr.com/ru/company/netologyru/blog/334914/>]

# Приложение

main.cpp

#include <iostream>

#include "Polinom.h"

using namespace std;

void main()

{

TList<double, unsigned int>\* l1 = new TList<double, unsigned int>();

l1->InsertEnd(4, -60);

l1->InsertEnd(43, -50);

l1->InsertEnd(326, 1.5);

TNode<double, unsigned int>\* node1 = new TNode<double, unsigned int>(200, 5);

TNode<double, unsigned int>\* node2 = new TNode<double, unsigned int>(17, 500);

cout << "node1 " << \*node1 << endl;

cout << "node2 " << \*node2 << endl;

Polinom p1(l1);

Polinom p2("-27.4x^2 + 5z^4 -2x^4y^4 +55z^4");

Polinom p3("15z^2 - 13y^2z^2");

Polinom p4;

cout << "p1 " << p1 << endl;

cout << "p2 " << p2 << endl;

cout << "p3 " << p3 << endl;

cout << "p4 is empty" << endl;

Polinom res;

cout << endl << "Tests" << endl;

try

{

cout << endl << "1) p1 + p2" << endl;

res = p1 + p2;

cout << "result:" << res << endl;

cout << "p4 + p3" << endl;

res = p4 + p3;

cout << "result:" << res << endl;

}

catch (const char\* str)

{

cout << str << endl;

}

try

{

cout << endl << "2) p1 - p2" << endl;

res = p1 - p2;

cout << "result:" << res << endl;

cout << "p4 - p3" << endl;

res = p4 - p3;

cout << "result:" << res << endl;

}

catch (const char\* str)

{

cout << str << endl;

}

try

{

cout << endl << "3) p2 \* p3" << endl;

res = p2 \* p3;

cout << "result:" << res << endl;

cout << "p1 \* p2" << endl;

res = p1 \* p2;

cout << "result:" << res << endl;

}

catch (const char\* str)

{

cout << str << endl;

}

try

{

cout << endl << "4) p1 += p2" << endl;

p1 += p2;

cout << "result:" << p1 << endl;

}

catch (const char\* str)

{

cout << str << endl;

}

try

{

cout << endl << "5) p1 -= p2" << endl;

p1 -= p2;

cout << "result:" << p1 << endl;

}

catch (const char\* str)

{

cout << str << endl;

}

try

{

cout << endl << "6) p2 + node1" << endl;

res = p2 + \*node1;

cout << "result:" << res << endl;

cout << "p2 + node2" << endl;

res = p2 + \*node2;

cout << "result:" << res << endl;

}

catch (const char\* str)

{

cout << str << endl;

}

try

{

cout << endl << "7) p2 - node1" << endl;

res = p2 - \*node1;

cout << "result:" << res << endl;

cout << "p2 - node2" << endl;

res = p2 - \*node2;

cout << "result:" << res << endl;

}

catch (const char\* str)

{

cout << str << endl;

}

try

{

cout << endl << "8) p2 \* node1" << endl;

res = p2 \* \*node1;

cout << "result:" << res << endl;

cout << "p2 \* node2" << endl;

res = p2 \* \*node2;

cout << "result:" << res << endl;

}

catch (const char\* str)

{

cout << str << endl;

}

try

{

cout << endl << "9) p1 += node1" << endl;

p1 += \*node1;

cout << "result:" << res << endl;

}

catch (const char\* str)

{

cout << str << endl;

}

try

{

cout << endl << "10) p1 -= node1" << endl;

p1 += \*node1;

cout << "result:" << res << endl;

}

catch (const char\* str)

{

cout << str << endl;

}

try

{

cout << endl << "11) p2 \*= node2" << endl;

p2 \*= \*node2;

cout << "result:" << p2 << endl;

}

catch (const char\* str)

{

cout << str << endl;

}

try

{

cout << endl << "12) p3 = -p3" << endl;

p3 = -p3;

cout << "result:" << p3 << endl;

}

catch (const char\* str)

{

cout << str << endl;

}

}

TNode.h

#ifndef TNODE\_H

#define TNODE\_H

#include <iostream>

using namespace std;

template<typename TData, typename TKey>

class TNode

{

public:

TData\* pData;

TKey key;

TNode\* pNext;

TNode();

TNode(TKey, TData\*, TNode\* \_pNext = nullptr);

TNode(const TNode&);

friend std::ostream& operator << (std::ostream& out, TNode<TData, TKey>& node)

{

out << node.key << " - " << node.pData;

return out;

}

};

#endif

TMonom.cpp

#include "TMonom.h"

TNode<double, unsigned int>::TNode() : pData(0), key(0), pNext(nullptr) {}

TNode<double, unsigned int>::TNode(unsigned int \_key, double \_pData, TNode\* \_pNext) : key(\_key), pData(\_pData), pNext(\_pNext) {}

TNode<double, unsigned int>::TNode(const TNode<double, unsigned int>& node) : key(node.key), pData(node.pData), pNext(nullptr) {}

TNode<double, unsigned int>& TNode<double, unsigned int>::operator=(const TNode& mon)

{

if (\*this == mon)

return \*this;

key = mon.key;

pData = mon.pData;

pNext = nullptr;

}

TNode<double, unsigned int> TNode<double, unsigned int>::operator+(const TNode<double, unsigned int>& mon)

{

if (key != mon.key)

throw "Monoms have diffrent degrees";

TNode<double, unsigned int> res(mon);

res.pData += pData;

return res;

}

TNode<double, unsigned int> TNode<double, unsigned int>::operator-(const TNode<double, unsigned int>& mon)

{

if (key != mon.key)

throw "Monoms have diffrent degrees";

TNode<double, unsigned int> res(\*this);

res.pData -= mon.pData;

return res;

}

TNode<double, unsigned int> TNode<double, unsigned int>::operator\*(const TNode<double, unsigned int>& mon)

{

if (((mon.key / 100) + (key / 100) > 9) || (((mon.key / 10) % 10) + ((key / 10) % 10) > 9) || ((mon.key % 10) + (key % 10) > 9))

throw "Degree limit exceeded";

TNode<double, unsigned int> res;

res.pData = pData \* mon.pData;

res.key = key + mon.key;

return res;

}

TNode<double, unsigned int> TNode<double, unsigned int>::operator\*(double a)

{

TNode<double, unsigned int> res(\*this);

res.pData \*= a;

return res;

}

TNode<double, unsigned int>& TNode<double, unsigned int>::operator+=(const TNode<double, unsigned int>& tmp)

{

if (key != tmp.key)

throw "Monoms have diffrent degrees";

pData += tmp.pData;

return \*this;

}

TNode<double, unsigned int>& TNode<double, unsigned int>::operator-=(const TNode<double, unsigned int>& tmp)

{

if (key != tmp.key)

throw "Monoms have diffrent degrees";

pData -= tmp.pData;

return \*this;

}

TNode<double, unsigned int>& TNode<double, unsigned int>::operator\*=(const TNode<double, unsigned int>& tmp)

{

if (((tmp.key / 100 + key / 100) > 9) || ((tmp.key / 10 + key / 10) > 90)

|| ((tmp.key % 10 + key % 10) > 9))

throw "Degree limit exceeded";

key += tmp.key;

pData \*= tmp.pData;

return \*this;

}

TNode<double, unsigned int>& TNode<double, unsigned int>::operator\*=(double c)

{

pData \*= c;

return \*this;

}

bool TNode<double, unsigned int>::operator==(const TNode<double, unsigned int>& mon) const

{

if ((key == mon.key) && (pData == mon.pData))

return true;

return false;

}

bool TNode<double, unsigned int>::operator!=(const TNode<double, unsigned int>& mon) const

{

return (!(\*this == mon));

}

TNode<double, unsigned int> TNode<double, unsigned int>::operator-() const

{

TNode<double, unsigned int> res(key, -pData);

return res;

}

TMonom.h

#ifndef TMONOM\_H

#define TMONOM\_H

#include "TNode.h"

#include <iostream>

using namespace std;

template<>

class TNode<double, unsigned int>

{

public:

double pData;

unsigned int key;

TNode\* pNext;

TNode();

TNode(unsigned int, double, TNode\* \_pNext = nullptr);

TNode(const TNode&);

TNode& operator=(const TNode&);

TNode operator+(const TNode&);

TNode operator-(const TNode&);

TNode operator\*(const TNode&);

TNode& operator+=(const TNode&);

TNode& operator-=(const TNode&);

TNode& operator\*=(const TNode&);

TNode& operator\*=(double);

TNode operator\*(double);

TNode operator-() const;

bool operator==(const TNode&) const;

bool operator!=(const TNode&) const;

friend ostream& operator<<(ostream& out, TNode& node)

{

if (node.pData > 0)

out << " +";

else

out << " ";

out << node.pData;

if (node.key / 100 != 0)

out << "x^" << node.key / 100;

if ((node.key / 10) % 10 != 0)

out << "y^" << (node.key / 10) % 10;

if (node.key % 10 != 0)

out << "z^" << node.key % 10;

return out;

}

};

#endif

TList.h

#ifndef TLIST\_H

#define TLIST\_H

#include "TMonom.h"

#include <iostream>

using namespace std;

template<typename TData, typename TKey>

class TList

{

private:

TNode<typename TData, typename TKey>\* pFirst;

TNode<typename TData, typename TKey>\* pPrev;

TNode<typename TData, typename TKey>\* pCur;

TNode<typename TData, typename TKey>\* pNext;

public:

TList();

TList(const TNode<TData, TKey>\*);

TList(const TList&);

~TList();

bool IsEnded() const;

void Next();

void Reset();

TNode<TData, TKey>\* Search(TKey);

void InsertEnd(TKey, TData\*);

void InsertFirst(TKey, TData\*);

void InsertAfter(TKey, TData\*, TKey);

void InsertBefore(TKey, TData\*, TKey);

void Remove(TKey);

friend std::ostream& operator << (std::ostream& out, TList<TData, TKey>& list)

{

list.Reset();

while (!(list.IsEnded()))

{

out << (\*list.pCur);

list.Next();

}

return out;

}

};

///////////////////////////////////////////////

template<typename TData, typename TKey>

TList<TData, TKey>::TList() :pFirst(0), pPrev(0), pCur(0), pNext(0) {}

template<typename TData, typename TKey>

TList<TData, TKey>::TList(const TNode<TData, TKey>\* \_node)

{

TNode<TData, TKey>\* node = \_node;

pFirst = new TNode<TData, TKey>(node->key, node->pData);

pCur = pFirst;

node = node->pNext;

while (node != nullptr)

{

pCur->pNext = new TNode<TData, TKey>(node->key, node->pData);

pCur = pCur->pNext;

node = node->pNext;

}

Reset();

}

template<typename TData, typename TKey>

TList<TData, TKey>::TList(const TList<TData, TKey>& list)

{

TNode<TKey, TData>\* node = list.pFirst;

pFirst = new TNode<TData, TKey>(node->key, node->pData);

pCur = pFirst;

node = node->pNext;

while (node != nullptr)

{

pCur->pNext = new TNode<TData, TKey>(node->key, node->pData);

pCur = pCur->pNext;

node = node->pNext;

}

Reset();

}

template<typename TData, typename TKey>

TList<TData, TKey>::~TList()

{

Reset();

while (pFirst != nullptr)

{

pNext = pFirst->pNext;

delete pFirst;

pFirst = pNext;

}

}

template<typename TData, typename TKey>

bool TList<TData, TKey>::IsEnded() const

{

return(pCur == nullptr);

}

template<typename TData, typename TKey>

void TList<TData, TKey>::Reset()

{

pPrev = nullptr;

pCur = pFirst;

if (pFirst != nullptr)

pNext = pCur->pNext;

else

pNext = nullptr;

}

template<typename TData, typename TKey>

void TList<TData, TKey>::Next()

{

if (IsEnded())

throw "You reached the end";

pPrev = pCur;

pCur = pNext;

if (pCur != nullptr)

pNext = pCur->pNext;

else

pNext = nullptr;

}

template<class TData, class TKey>

TNode<TData, TKey>\* TList<TData, TKey>::Search(TKey fkey)

{

Reset();

while ((!IsEnded()) && (pCur->key != fkey))

Next();

if (IsEnded())

{

Reset();

return nullptr;

}

return pCur;

}

template<typename TData, typename TKey>

void TList<TData, TKey>::InsertEnd(TKey \_key, TData\* \_pData) //v konec

{

if (pFirst == nullptr)

{

pFirst = new TNode<TData, TKey>(\_key, \_pData);

pCur = pFirst;

return;

}

TNode<TData, TKey>\* fpPrev = pPrev;

TNode<TData, TKey>\* fpCur = pCur;

TNode<TData, TKey>\* fpNext = pNext;

Reset();

while (!IsEnded())

Next();

TNode<TData, TKey>\* node = new TNode<TData, TKey>(\_key, \_pData);

pPrev->pNext = node;

pPrev = fpPrev;

pCur = fpCur;

if (pCur->pNext == nullptr)

pNext = node;

else pNext = fpNext;

}

template<class TData, class TKey>

void TList<TData, TKey>::InsertFirst(TKey \_key, TData\* \_pData)// v nachalo

{

TNode<TData, TKey>\* node = new TNode<TData, TKey>(\_key, \_pData, pFirst);

if (pCur == pFirst)

{

pPrev = node;

}

pFirst = node;

}

template<class TData, class TKey>

void TList<TData, TKey>::InsertAfter(TKey fkey, TData\* \_pData, TKey \_key)

{

if (pFirst == 0)

throw "The list is empty";

TNode<TData, TKey>\* fpPrev = pPrev;

TNode<TData, TKey>\* fpCur = pCur;

TNode<TData, TKey>\* fpNext = pNext;

Reset();

while (!IsEnded())

{

if (pCur->key == fkey)

{

TNode<TData, TKey>\* node = new TNode<TData, TKey>(\_key, \_pData, pNext);

pCur->pNext = node;

pPrev = fpPrev;

pCur = fpCur;

if (pCur != fpCur)

{

pNext = fpNext;

}

else

{

pNext = node;

}

return;

}

Next();

}

pPrev = fpPrev;

pCur = fpCur;

pNext = fpNext;

throw "The list doesn't contain an elem with this key";

}

template<class TData, class TKey>

void TList<TData, TKey>::InsertBefore(TKey fkey, TData\* \_pData, TKey \_key)

{

if (pFirst == 0)

throw "The list is empty";

TNode<TData, TKey>\* fpFirst = pFirst;

TNode<TData, TKey>\* fpPrev = pPrev;

TNode<TData, TKey>\* fpCur = pCur;

TNode<TData, TKey>\* fpNext = pNext;

Reset();

if (pCur->key == fkey)

{

TNode<TData, TKey>\* node = new TNode<TData, TKey>(\_key, \_pData, pCur);

pFirst = node;

if (fpCur == fpFirst)

pPrev = node;

else

pPrev = fpPrev;////

pCur = fpCur;

pNext = fpNext;

return;

}

while (pCur->pNext != nullptr)

{

if (pNext->key == fkey)

{

TNode<TData, TKey>\* node = new TNode<TData, TKey>(\_key, \_pData, pNext);

pCur->pNext = node;

pPrev = fpPrev;

pCur = fpCur;

if (pCur != fpCur)

{

pNext = fpNext;

}

else

{

pNext = node;

}

return;

}

Next();

}

pPrev = fpPrev;

pCur = fpCur;

pNext = fpNext;

throw "The list doesn't contain an elem with this key";

}

template<class TData, class TKey>

void TList<TData, TKey>::Remove(TKey fkey)

{

if (pFirst == nullptr)

throw "The list is empty";

TNode<TData, TKey>\* fpFirst = pFirst;

TNode<TData, TKey>\* fpPrev = pPrev;

TNode<TData, TKey>\* fpCur = pCur;

TNode<TData, TKey>\* fpNext = pNext;

if (Search(fkey) == nullptr)

{

pPrev = fpPrev;

pCur = fpCur;

pNext = fpNext;

throw "The list doesn't contain an elem with this key";

}

if (pFirst == pCur)

{

if (fpCur == pFirst)

{

Next();

fpCur = pCur;

fpNext = pNext;

fpPrev->pNext = pCur;

fpFirst = pCur;

}

else if (fpCur == pFirst->pNext)

{

fpPrev = nullptr;

fpPrev->pNext = fpCur;

fpFirst = fpCur;

}

else

{

fpFirst = fpFirst->pNext;

}

delete pFirst;

pFirst = fpFirst;

pPrev = fpPrev;

pCur = fpCur;

pNext = fpNext;

return;

}

if (fpCur == pCur)

{

delete pCur;

pCur = fpNext;

pPrev->pNext = pCur;

if (pNext != nullptr)

{

pNext = pNext->pNext;

}

return;

}

pPrev->pNext = pCur->pNext;

delete pCur;

pCur = pNext;

if (pNext != nullptr)

{

pNext = pNext->pNext;

}

pPrev = fpPrev;

pCur = fpCur;

pNext = fpNext;

}

#endif

TSList.cpp

#include "TSList.h"

#include <iostream>

using namespace std;

TList<double, unsigned int>::TList() :pFirst(0), pPrev(0), pCur(0), pNext(0) {}

TList<double, unsigned int>::TList(const TNode<double, unsigned int>\* \_node)

{

TNode<double, unsigned int>\* node = new TNode<double, unsigned int>(\*\_node);

pFirst = new TNode<double, unsigned int>(node->key, node->pData);

pCur = pFirst;

node = node->pNext;

while (node != nullptr)

{

pCur->pNext = new TNode<double, unsigned int>(node->key, node->pData);

pCur = pCur->pNext;

node = node->pNext;

}

Reset();

}

TList<double, unsigned int>::TList(const TList<double, unsigned int>& list)

{

if (list.pFirst == nullptr)

{

TList();

return;

}

TNode<double, unsigned int>\* node = list.pFirst;

pFirst = new TNode<double, unsigned int>(node->key, node->pData);

pCur = pFirst;

node = node->pNext;

while (node != nullptr)

{

pCur->pNext = new TNode<double, unsigned int>(node->key, node->pData);

pCur = pCur->pNext;

node = node->pNext;

}

Reset();

}

TList<double, unsigned int>::~TList()

{

Reset();

while (pFirst != nullptr)

{

pNext = pFirst->pNext;

delete pFirst;

pFirst = pNext;

}

}

bool TList<double, unsigned int>::operator==(const TList& tmp) const

{

TNode<double, unsigned int>\* node1 = pFirst;

TNode<double, unsigned int>\* node2 = tmp.pFirst;

while (node1 && node2 && (\*node1 == \*node2))

{

node1 = node1->pNext;

node2 = node2->pNext;

}

return (!(node1 || node2));

}

bool TList<double, unsigned int>::IsEnded() const

{

return(pCur == nullptr);

}

void TList<double, unsigned int>::Reset()

{

pPrev = nullptr;

pCur = pFirst;

if (pFirst != nullptr)

pNext = pCur->pNext;

else

pNext = nullptr;

}

void TList<double, unsigned int>::Next()

{

if (IsEnded())

throw "You reached the end";

pPrev = pCur;

pCur = pNext;

if (pCur != nullptr)

pNext = pCur->pNext;

else

pNext = nullptr;

}

TNode<double, unsigned int>\* TList<double, unsigned int>::Search(unsigned int fkey)

{

Reset();

while ((!IsEnded()) && (pCur->key != fkey))

Next();

if (IsEnded())

{

Reset();

return nullptr;

}

return pCur;

}

void TList<double, unsigned int>::InsertEnd(unsigned int \_key, double \_pData) //v konec

{

if (pFirst == nullptr)

{

pFirst = new TNode<double, unsigned int>(\_key, \_pData);

pCur = pFirst;

return;

}

TNode<double, unsigned int>\* fpPrev = pPrev;

TNode<double, unsigned int>\* fpCur = pCur;

TNode<double, unsigned int>\* fpNext = pNext;

Reset();

while (!IsEnded())

Next();

TNode<double, unsigned int>\* node = new TNode<double, unsigned int>(\_key, \_pData);

pPrev->pNext = node;

pPrev = fpPrev;

pCur = fpCur;

if (pCur->pNext == nullptr)

pNext = node;

else pNext = fpNext;

}

void TList<double, unsigned int>::InsertFirst(unsigned int \_key, double \_pData)// v nachalo

{

TNode<double, unsigned int>\* node = new TNode<double, unsigned int>(\_key, \_pData, pFirst);

if (pCur == pFirst)

{

pPrev = node;

}

pFirst = node;

}

void TList<double, unsigned int>::InsertAfter(unsigned int fkey, double \_pData, unsigned int \_key)

{

if (pFirst == 0)

throw "The list is empty";

TNode<double, unsigned int>\* fpPrev = pPrev;

TNode<double, unsigned int>\* fpCur = pCur;

TNode<double, unsigned int>\* fpNext = pNext;

Reset();

while (!IsEnded())

{

if (pCur->key == fkey)

{

TNode<double, unsigned int>\* node = new TNode<double, unsigned int>(\_key, \_pData, pNext);

pCur->pNext = node;

pPrev = fpPrev;

pCur = fpCur;

if (pCur != fpCur)

{

pNext = fpNext;

}

else

{

pNext = node;

}

return;

}

Next();

}

pPrev = fpPrev;

pCur = fpCur;

pNext = fpNext;

throw "The list doesn't contain an elem with this key";

}

void TList<double, unsigned int>::InsertBefore(unsigned int fkey, double \_pData, unsigned int \_key)

{

if (pFirst == 0)

throw "The list is empty";

TNode<double, unsigned int>\* fpFirst = pFirst;

TNode<double, unsigned int>\* fpPrev = pPrev;

TNode<double, unsigned int>\* fpCur = pCur;

TNode<double, unsigned int>\* fpNext = pNext;

Reset();

if (pCur->key == fkey)

{

InsertFirst(\_key, \_pData);

return;

}

while (pCur->pNext != nullptr)

{

if (pNext->key == fkey)

{

TNode<double, unsigned int>\* node = new TNode<double, unsigned int>(\_key, \_pData, pNext);

pCur->pNext = node;

pPrev = fpPrev;

pCur = fpCur;

if (pCur != fpCur)

{

pNext = fpNext;

}

else

{

pNext = node;

}

return;

}

Next();

}

pPrev = fpPrev;

pCur = fpCur;

pNext = fpNext;

throw "The list doesn't contain an elem with this key";

}

void TList<double, unsigned int>::Remove(unsigned int fkey)

{

if (pFirst == nullptr)

throw "The list is empty";

TNode<double, unsigned int>\* fpFirst = pFirst;

TNode<double, unsigned int>\* fpPrev = pPrev;

TNode<double, unsigned int>\* fpCur = pCur;

TNode<double, unsigned int>\* fpNext = pNext;

if (Search(fkey) == nullptr)

{

pPrev = fpPrev;

pCur = fpCur;

pNext = fpNext;

throw "The list doesn't contain an elem with this key";

}

if (pFirst == pCur)

{

if (fpCur == pFirst)

{

Next();

fpCur = pCur;

fpNext = pNext;

fpPrev->pNext = pCur;

fpFirst = pCur;

}

else if (fpCur == pFirst->pNext)

{

fpPrev = nullptr;

fpPrev->pNext = fpCur;

fpFirst = fpCur;

}

else

{

fpFirst = fpFirst->pNext;

}

delete pFirst;

pFirst = fpFirst;

pPrev = fpPrev;

pCur = fpCur;

pNext = fpNext;

return;

}

if (fpCur == pCur)

{

delete pCur;

pCur = fpNext;

pPrev->pNext = pCur;

if (pNext != nullptr)

{

pNext = pNext->pNext;

}

return;

}

pPrev->pNext = pCur->pNext;

delete pCur;

pCur = pNext;

if (pNext != nullptr)

{

pNext = pNext->pNext;

}

pPrev = fpPrev;

pCur = fpCur;

pNext = fpNext;

}

TSList.h

#ifndef TSLIST\_H

#define TSLIST\_H

#include "TNode.h"

#include "TList.h"

#include "TMonom.h"

#include <iostream>

using namespace std;

template<>

class TList<double, unsigned int>

{

public:

TNode<double, unsigned int>\* pFirst;

TNode<double, unsigned int>\* pPrev;

TNode<double, unsigned int>\* pCur;

TNode<double, unsigned int>\* pNext;

TList();

TList(const TNode<double, unsigned int>\*);

TList(const TList&);

~TList();

bool operator==(const TList&) const;

bool IsEnded() const;

void Next();

void Reset();

TNode<double, unsigned int>\* Search(unsigned int);

void InsertEnd(unsigned int, double);

void InsertFirst(unsigned int, double);

void InsertAfter(unsigned int, double, unsigned int);

void InsertBefore(unsigned int, double, unsigned int);

void Remove(unsigned int);

friend std::ostream& operator << (std::ostream& out, TList<double, unsigned int>& list)

{

list.Reset();

while (!(list.IsEnded()))

{

out << (\*list.pCur);

list.Next();

}

return out;

}

};

#endif

Polinom.cpp

#include "Polinom.h"

#include <iostream>

#include <string>

using namespace std;

Polinom::Polinom()

{

pol = new TList<double, unsigned int>();

}

Polinom::Polinom(const string& str)

{

pol = new TList<double, unsigned int>();

int \_key = 0;

double pc = 1;

int dx = 0, dy = 0, dz = 0;

bool flag = false;

char c = '+';

for (int i = 0; i <= str.length(); i++)

{

int j = 0;

char coef[10];

if (((str[i] == '+') || (str[i] == '-')) && (flag == false))

{

flag = true;

c = str[i];

i++;

}

if (((str[i] == '+') || (str[i] == '-') || (i == str.length())) && (flag == true))

{

\_key = dx \* 100 + dy \* 10 + dz;

if (c == '-')

pol->InsertEnd(\_key, -pc);

if (c == '+')

pol->InsertEnd(\_key, pc);

if (!(i == str.length()))

c = str[i];

pc = 1;

dx = 0, dy = 0, dz = 0;

}

if (isdigit(str[i]))

{

flag = true;

while ((isdigit(str[i + j])) || (str[i + j] == '.'))

{

coef[j] = str[i + j];

j++;

}

i += j;

pc = (float)(atof(coef));

memset(coef, 0, sizeof(coef));

}

if (str[i] == 'x')

{

flag = true;

if (str[i + 1] == '^')

{

dx = static\_cast<int>(str[i + 2]) - '0';

i += 2;

}

else

dx = 1;

}

if (str[i] == 'y')

{

flag = true;

if (str[i + 1] == '^')

{

dy = static\_cast<int>(str[i + 2]) - '0';

i += 2;

}

else

dy = 1;

}

if (str[i] == 'z')

{

flag = true;

if (str[i + 1] == '^')

{

dz = static\_cast<int>(str[i + 2]) - '0';

i += 2;

}

else

dz = 1;

}

}

pol = StandartView();

}

Polinom::Polinom(TList<double, unsigned int>\* tmp)

{

pol = new TList<double, unsigned int>(\*tmp);

pol = StandartView();

}

Polinom::Polinom(const Polinom& tmp)

{

pol = new TList<double, unsigned int>(\*tmp.pol);

}

Polinom::~Polinom()

{

delete pol;

}

TList<double, unsigned int>\* Polinom::StandartView()

{

if ((pol->pFirst == nullptr) || (pol->pFirst->pNext == nullptr))

return pol;

TList<double, unsigned int>\* res = new TList<double, unsigned int>();

bool flag = false;

pol->Reset();

res->InsertEnd(pol->pCur->key, pol->pCur->pData);

pol->Next();

while (!pol->IsEnded())

{

res->Reset();

while ((!res->IsEnded()) && (flag == false))

{

if (pol->pCur->key > res->pCur->key)

{

res->InsertBefore(res->pCur->key, pol->pCur->pData, pol->pCur->key);

flag = true;

}

res->Next();

}

if (pol->pCur->key < res->pPrev->key)

res->InsertAfter(res->pPrev->key, pol->pCur->pData, pol->pCur->key);

if (pol->pCur->key == res->pPrev->key)

res->pPrev->pData += pol->pCur->pData;

pol->Next();

flag = false;

}

return res;

}

bool Polinom::operator==(const Polinom& tmp) const

{

if (\*this->pol == \*tmp.pol)

return true;

}

const Polinom& Polinom::operator=(const Polinom& tmp)

{

if (\*this == tmp)

return \*this;

delete pol;

pol = new TList<double, unsigned int>(\*tmp.pol);

return \*this;

}

Polinom Polinom::operator+(const Polinom& p)

{

Polinom res;

TNode<double, unsigned int>\* tmp1 = pol->pFirst;

TNode<double, unsigned int>\* tmp2 = p.pol->pFirst;

while ((tmp1 != nullptr) && (tmp2 != nullptr))

{

if (tmp1->key > tmp2->key)

{

res.pol->InsertEnd(tmp1->key, tmp1->pData);

tmp1 = tmp1->pNext;

}

else if (tmp1->key < tmp2->key)

{

res.pol->InsertEnd(tmp2->key, tmp2->pData);

tmp2 = tmp2->pNext;

}

else

{

double t = tmp1->pData + tmp2->pData;

if (t != 0)

{

res.pol->InsertEnd(tmp1->key, t);

}

tmp1 = tmp1->pNext;

tmp2 = tmp2->pNext;

}

}

if (tmp1 == nullptr)

{

while (tmp2 != nullptr)

{

res.pol->InsertEnd(tmp2->key, tmp2->pData);

tmp2 = tmp2->pNext;

}

}

else if (tmp2 == nullptr)

{

while (tmp2 != nullptr)

{

res.pol->InsertEnd(tmp1->key, tmp1->pData);

tmp1 = tmp1->pNext;

}

}

return res;

}

Polinom& Polinom::operator+=(const Polinom& tmp)

{

Polinom p = \*this;

\*this = p + tmp;

return \*this;

}

Polinom Polinom::operator-(const Polinom& tmp)

{

Polinom res(tmp);

res = -res;

return (\*this + (res));

}

Polinom& Polinom::operator-=(const Polinom& tmp)

{

Polinom p = \*this;

\*this = p - tmp;

return \*this;

}

Polinom Polinom::operator\*(const Polinom& p)

{

if((pol->pFirst == nullptr) || (p.pol->pFirst == nullptr))

return 0;

Polinom tmp1(p);

Polinom tmp2(\*this);

tmp1.pol->Reset();

tmp2.pol->Reset();

Polinom res;

while (!tmp1.pol->IsEnded())

{

res += tmp2 \* \*tmp1.pol->pCur;

tmp1.pol->Next();

}

return res;

}

Polinom Polinom::operator+(const TNode<double, unsigned int>& node)

{

if (node.pData == 0)

return \*this;

Polinom res(\*this);

TNode<double, unsigned int>\* tmp = res.pol->pFirst;

while ((tmp != nullptr) && (node.key < tmp->key))

tmp = tmp->pNext;

if (tmp == nullptr)

{

res.pol->InsertEnd(node.key, node.pData);

return res;

}

if (tmp->key == node.key)

{

if (tmp->pData + node.pData != 0)

{

tmp->pData += node.pData;

return res;

}

res.pol->Remove(tmp->key);

}

res.pol->InsertBefore(tmp->key, node.pData, node.key);

return res;

}

Polinom Polinom::operator-(const TNode<double, unsigned int>& node)

{

Polinom res = \*this + (-node);

return res;

}

Polinom Polinom::operator\*(const TNode<double, unsigned int>& node)

{

if (node.pData == 0)

return 0;

Polinom res(\*this);

res.pol->Reset();

while (!res.pol->IsEnded())

{

\*res.pol->pCur \*= node;

res.pol->Next();

}

return res;

}

Polinom& Polinom::operator+=(const TNode<double, unsigned int>& node)

{

Polinom tmp = (\*this) + node;

\*this = tmp;

return \*this;

}

Polinom& Polinom::operator-=(const TNode<double, unsigned int>& node)

{

Polinom tmp = (\*this) - node;

\*this = tmp;

return \*this;

}

Polinom& Polinom::operator\*=(const TNode<double, unsigned int>& node)

{

Polinom tmp = (\*this) \* node;

\*this = tmp;

return \*this;

}

Polinom Polinom::operator-()

{

Polinom tmp(\*this);

while (!tmp.pol->IsEnded())

{

tmp.pol->pCur->pData \*= (-1);

tmp.pol->Next();

}

tmp.pol->Reset();

return tmp;

}

Polinom.h

#ifndef POLINOM\_H

#define POLINOM\_H

#include "TSList.h"

#include <iostream>

using namespace std;

class Polinom

{

private:

TList<double, unsigned int>\* pol;

public:

Polinom();

Polinom(const string&);

Polinom(TList<double, unsigned int>\*);

Polinom(const Polinom&);

~Polinom();

TList<double, unsigned int>\* StandartView();

bool operator==(const Polinom&) const;

const Polinom& operator=(const Polinom&);

Polinom operator+(const Polinom&);

Polinom& operator+=(const Polinom&);

Polinom operator-(const Polinom&);

Polinom& operator-=(const Polinom&);

Polinom operator\*(const Polinom&);

Polinom operator+(const TNode<double, unsigned int>&);

Polinom operator-(const TNode<double, unsigned int>&);

Polinom operator\*(const TNode<double, unsigned int>&);

Polinom& operator+=(const TNode<double, unsigned int>&);

Polinom& operator-=(const TNode<double, unsigned int>&);

Polinom& operator\*=(const TNode<double, unsigned int>&);

Polinom operator-();

friend ostream& operator<<(ostream& out, Polinom& tmp)

{

out << \*tmp.pol;

return out;

}

};

#endif