Министерство науки и высшего образования Российской Федерации   
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:

Студент группы 381806-3

Макарычев С.Д.

Проверил:

к.т.н., доцент кафедры МОСТ ИИТММ

Кустикова В.Д.

Нижний Новгород

2019 г.

[Введение 3](#_Toc36122756)

[1 Постановка задачи 4](#_Toc36122757)

[2 Руководство пользователя 5](#_Toc36122758)

[3 Руководство программиста 7](#_Toc36122759)

[3.1 Структура программы 7](#_Toc36122760)

[3.2 Описание структур данных 7](#_Toc36122761)

[3.2.1 Класс TNode 7](#_Toc36122762)

[3.2.2 Класс TMonom 7](#_Toc36122763)

[3.2.3 Класс TList 7](#_Toc36122764)

[3.2.4 Класс Polinom 10](#_Toc36122765)

[3.2.5 Класс M\_Exeption 10](#_Toc36122766)

[3.3 Описание алгоритмов 11](#_Toc36122767)

[3.3.1 Алгоритмы работы со списком 11](#_Toc36122768)

[3.3.2 Алгоритмы работы с полиномом 11](#_Toc36122769)

[Вывод 13](#_Toc36122770)

[Приложение 14](#_Toc36122771)

# Введение

Вместе с вычислительным применением компьютеры используются также и для аналитической обработки данных. Среди примеров такого применения присутствует задача обработки полиномов, задаваемых в общей аналитической форме.

В данной работе представлена реализация структуры хранения полиномов с помощью линейных односвязных списков, а также алгоритмов, предназначенных для выполнения операций над полиномами.

# Постановка задачи

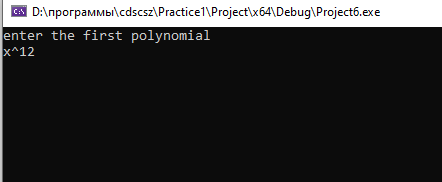
В данной работе необходимо, используя структуру данных “Линейные односвязные списки”, реализовать программу, позволяющую преобразовать строку с полиномом в звенья односвязного списка и выполнять операции над этим полиномом.

Входные данные: строки, содержащие полиномы; операции, проводимые над этими полиномами.

Выходные данные: строка, содержащая результат операции – полином.

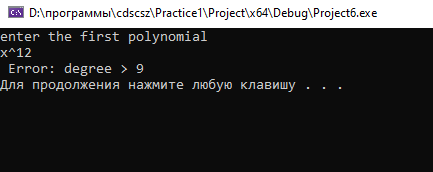
# Руководство пользователя

После запуска программы пользователю необходимо ввести первый полином с положительными и не превосходящими девяти степенями. Вводим в поле наш полином и нажимаем Enter (Рис. 1).



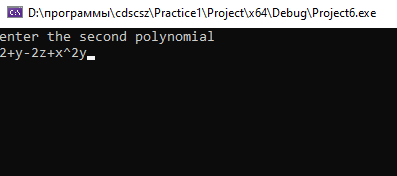
1. Ввод первого полинома.

Если полином введен не корректно, степень какой-то из переменных больше девяти или меньше нуля, то сработает исключительная ситуация (Рис. 2).



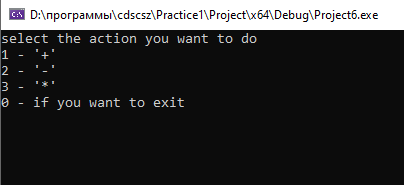
1. Исключительная ситуация: некорректный ввод.

Если же ввод был произведён корректно, то дальше нужно будет ввести второй полином. Вводим в поле второй полином и нажимаем Enter (Рис. 3).



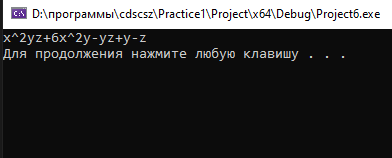
1. Ввод второго полинома.

Далее пользователю предлагается выбрать действие, которое он хочет выполнить над введёнными полиномами (Рис. 4).



1. Выбор операции над полиномами.

После выбора необходимого действия в консоль выведется результат (Рис. 5).



1. Вывод результата на экран.

# Руководство программиста

## Структура программы

Данная программа состоит из 6 файлов: TNode.h с шаблонным классом – реализацией звена линейного списка; файла TMonom.h, содержащий специализацию шаблонного класса; файла TList.h, в котром описан шаблонный класс, реализующий структуру данных “Линейный односвязный список”; файла Polinom.h с классом, реализующим операции над полиномами; файла Exception.h с классом для обработки исключений; файла main.cpp, в котором реализуется основная программа.

## Описание структур данных

### Класс TNode

**Поля:**

TKey key

Задает идентификатор звена.

TData pData

Задаёт поле данных, хранимых в звене.

TNode<TKey, TData>\* pNext

Хранит указатель на следующее звено.

### Класс TMonom

**Поля:**

Unsigned int key

Задает идентификатор звена – степень монома.

float pData

Задаёт поле данных, хранимых в звене. Коэффициент монома.

TNode<unsigned int, float>\* pNext

Хранит указатель на следующий моном.

### Класс TList

**Поля:**

TNode<TKey, TData>\* pFirst

Хранит указатель на первый элемент списка.

TNode<TKey, TData>\* pCurrent

Хранит указатель на текущий элемент списка.

TNode<TKey, TData>\* pPrevious

Хранит указатель на предыдущий перед текущим элемент списка.

TNode<TKey, TData>\* pNext

Хранит указатель на следующий за текущим элемент списка.

**Методы:**

Void Sort()

Описание: сортирует список в порядке убывания ключей.

Ничего не принимает. Ничего не возвращает.

Void BringingSimilar()

Описание: объединяет данные звеньев с одинаковыми ключами.

Ничего не принимает. Ничего не возвращает.

Void InsertBegin(TKey,TData)

Описание: создаёт звено с заданным ключом и данными, вставляет его в начало списка.

Принимает ключ и данные. Ничего не возвращает.

Void InsertBegin(TNode<TKey,TData>\*)

Описание: вставляет данное звено в начало списка.

Принимает указатель на звено. Ничего не возвращает.

Void InsertEnd(TKey,TData)

Описание: создаёт звено с заданным ключом и данными, вставляет его в конец списка.

Принимает ключ и данные. Ничего не возвращает.

Void InsertEnd(TNode<TKey,TData>\*)

Описание: вставляет данное звено в конец списка.

Принимает указатель на звено. Ничего не возвращает.

Void InsertAfter(TKey, TNode<TKey,TData>\*)

Описание: вставляет звено после заданного ключа.

Принимает ключ и указатель на звено. Ничего не возвращает.

Void InsertBefore(TKey, TNode<TKey,TData>\*)

Описание: вставляет звено перед заданным ключом.

Принимает ключ и указатель на звено. Ничего не возвращает.

Void Remove(TKey)

Описание: удаляет звено с заданным ключом.

Принимает ключ удаляемого звена. Ничего не возвращает.

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

Описание: ищет звено с заданным ключом.

Принимает ключ звена, которое нужно найти. Возвращает звено с заданным ключом.

boll IsEmpty()const

Описание: константный метод, проверяющий есть ли в списке звенья.

Ничего не принимает. Возвращает ноль, если список не пуст, единицу, если пуст.

boll IsEnded()const

Описание: константный метод, проверяющий полон ли список.

Ничего не принимает. Возвращает ноль, если список не полон, единицу, если полон.

void Reset()

Описание: ставит указатель на текущий элемент в начало списка.

Ничего не принимает. Ничего не возвращает.

void Next()

Описание: смещает указатели на текущий, следующий и предыдущий на один вперёд, если это возможно.

Ничего не принимает. Ничего не возвращает.

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

Описание: константный метод, возвращающий указатель на первый элемент списка.

Ничего не принимает. Возвращает указатель на первый элемент списка.

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

Описание: константный метод, возвращающий указатель на текущий элемент списка.

Ничего не принимает. Возвращает указатель на текущий элемент списка.

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

Описание: константный метод, возвращающий указатель на предшествующий текущему элемент списка.

Ничего не принимает. Возвращает указатель на предшествующий текущему элемент списка.

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

Описание: константный метод, возвращающий указатель на следующий за текущим элемент списка.

Ничего не принимает. Возвращает указатель на следующий за текущим элемент списка.

### Класс Polinom

**Поля:**

TList<unsigned int, float>\* monoms

Массив звеньев списка.

**Методы:**

void SortMonoms()

Описание: сортирует мономы в порядке убывания степеней.

Ничего не принимает. Ничего не возвращает.

void SimillarMonoms()

Описание: приводит подобные мономы.

Ничего не принимает. Ничего не возвращает.

### Класс M\_Exeption

const char\* what() const noexcept

Описание: Передаёт строку с ошибкой

Ничего не принимает, возвращает строку с описанием ошибки.

## Описание алгоритмов

### Алгоритмы работы со списком

В программе мы используем структуру данных линейный односвязный список, которая предоставляет последовательный доступ к звеньям списка.

Основные характеристики линейного односвязного списка:

1. Указатель на первое звено списка.
2. Указатель на текущее звено списка.
3. Указатель на следующее звено списка.
4. Указатель на предыдущее звено списка.

Операции над списком:

1. Вставка элемента в начало списка.
2. Вставка элемента в конец списка.
3. Вставка до элемента с заданным ключом.
4. Вставка элемента после элемента с заданным ключом.
5. Удаление элемента с заданным ключом.
6. Поиск элемента с заданным ключом.
7. Методы навигации по списку.

### Алгоритмы работы с полиномом

1. Сложение полинома и монома:
2. Идём по полиному, пока не дойдем до звена, степень которого меньше степени прибавляемого монома.
3. Если мы дошли до конца полинома, то делаем вставку монома в конец полинома.
4. Если же мы нашли звено с меньшей степенью, чем у прибавляемого монома, то делаем вставку этого монома перед текущим звеном.
5. Если степень монома равна степени текущего звена и сумма их коэффициентов не равна нулю, то делаем вставку нового звена с коэффициентом, равным сумме коэффициентов текущего звена и монома после текущего звена, удаляем текущее звено. Если же сумма их коэффициентов равна нулю, то удаляем текущее звено.
6. Вычитание монома из полинома:

Происходит аналогично сложению полинома и монома.

1. Сложение двух полиномов:
2. Создаём полином, который будет равен сумме дух исходных полиномов.
3. Идём по звеньям этих полиномов, пока не дойдём до конца одного из них.
4. Если степень звена первого полинома больше степени звена второго полинома, то прибавляем звено первого полинома к сумме.
5. Если же степень звена первого полинома меньше степени звена второго полинома, то прибавляем звено второго полинома к сумме.
6. Если степени звена первого и второго полиномов равны, то если сумма их коэффициентов не равна нулю, прибавляем к сумме полиномов моном, равный сумме этих мономов. В противном случае, если сумма коэффициентов равна нулю, то ничего не прибавляем, а сдвигаемся на следующее звено в обоих полиномах.
7. Дойдя до конца одного из полиномов прибавляем оставшиеся мономы второго полинома к сумме полиномов.
8. Вычитание полиномов:

Производится аналогично сложению полиномов.

1. Умножение полинома на моном:

Идём по полиному и умножаем каждое звено на моном, произведение двух мономов добавляем в новый полином.

1. Умножение полинома на полином:

Идём по звеньям второго полинома и умножаем каждое из них на первый полином.

# Вывод

В ходе выполнения данной работы я ознакомился с такой структурой данных, как линейный односвязный список, и сделал вывод, что такой способ хранения в памяти наиболее всего подходит для полиномов. Также я изучил алгоритмы работы со списками и алгоритмы операций над полиномами.

# Приложение

Файл TNode.h

#ifndef TNODE\_H

#define TNODE\_H

using namespace std;

template<class TKey, class TData>

class TNode

{

public:

TKey key;

TData pData;

TNode<TKey, TData>\* pNext;

TNode();

TNode(TKey, TData, TNode<TKey, TData>\* pNext = 0);

TNode(const TNode<TKey, TData>&);

~TNode();

};

template<class TKey, class TData>

TNode<TKey, TData>::TNode()

{

key = 0;

pNext = nullptr;

}

template<class TKey, class TData>

TNode<TKey, TData>::TNode(TKey key, TData pData, TNode<TKey, TData>\* pNext)

{

this->key = key;

this->pData = pData;

this->pNext = pNext;

}

template<class TKey, class TData>

TNode<TKey, TData>::TNode(const TNode<TKey, TData>& node)

{

this->key = node.key;

this->pData = node.pData;

pNext = nullptr;

}

template<class TKey, class TData>

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

{

pNext = nullptr;

}

#endif

Файл TMonom.h

#ifndef TMONOM\_H

#define TMONOM\_H

#include "TNode.h"

#include "Exception.h"

template<>

class TNode<unsigned int, float>

{

public:

unsigned int key;

float pData;

TNode<unsigned int, float>\* pNext;

TNode() : key(0), pData(0), pNext(nullptr)

{

}

TNode(int NewKey, float Data, TNode<unsigned int, float>\* next = nullptr) :key(NewKey), pData(Data), pNext(next)

{

}

TNode(const TNode& tmp) : key(tmp.key), pData(tmp.pData), pNext(nullptr)

{

}

TNode(unsigned int deg\_x, unsigned int deg\_y, unsigned int deg\_z, float coeff, TNode<unsigned int, float>\* next = nullptr);

~TNode()

{

pNext = nullptr;

}

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

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

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

TNode<unsigned int, float>\* operator-()const;

bool operator <(const TNode<unsigned int, float>&)const;

bool operator >(const TNode<unsigned int, float>&)const;

bool operator !=(const TNode<unsigned int, float>&)const;

bool operator ==(const TNode<unsigned int, float>&)const;

const TNode<unsigned int, float>& operator= (const TNode<unsigned int, float>& monom);

};

TNode<unsigned int, float> ::TNode(unsigned int deg\_x, unsigned int deg\_y, unsigned int deg\_z, float coeff, TNode<unsigned int, float>\* next)

{

if (deg\_x > 9 || deg\_x < 0 || deg\_y > 9 || deg\_y < 0 || deg\_z > 9 || deg\_z < 0)

throw M\_Exeption("invalid parameter");

this->key = deg\_x \* 100 + deg\_y \* 10 + deg\_z;

this->pData = coeff;

this->pNext = next;

}

bool TNode<unsigned int, float> ::operator <(const TNode& OtherMonom)const

{

return(this->key < OtherMonom.key);

}

bool TNode<unsigned int, float> ::operator >(const TNode& OtherMonom)const

{

return(this->key > OtherMonom.key);

}

bool TNode<unsigned int, float> ::operator ==(const TNode& OtherMonom)const

{

return(this->key == OtherMonom.key && this->pData == OtherMonom.pData);

}

bool TNode<unsigned int, float> ::operator !=(const TNode& OtherMonom)const

{

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

}

const TNode<unsigned int, float>& TNode<unsigned int, float> :: operator=(const TNode<unsigned int, float>& monom)

{

if (\*this == monom)

return \*this;

this->key = monom.key;

this->pData = monom.pData;

return \*this;

}

TNode<unsigned int, float> TNode<unsigned int, float>::operator\*(const TNode<unsigned int, float>& monom)

{

int deg\_x1 = this->key / 100;

int deg\_y1 = (this->key - deg\_x1 \* 100) / 10;

int deg\_z1 = this->key - deg\_x1 \* 100 - deg\_y1 \* 10;

int deg\_x2 = monom.key / 100;

int deg\_y2 = (monom.key - deg\_x2 \* 100) / 10;

int deg\_z2 = monom.key - deg\_x2 \* 100 - deg\_y2 \* 10;

if (deg\_z1 + deg\_z2 > 9 || deg\_y1 + deg\_y2 > 9 || deg\_x1 + deg\_x2 > 9)

throw M\_Exeption("invalid parameter");

TNode<unsigned int, float> product(this->key + monom.key, this->pData \* monom.pData);

return product;

}

TNode<unsigned int, float> TNode<unsigned int, float>::operator+(const TNode<unsigned int, float>& monom)

{

if (this->key != monom.key)

throw M\_Exeption("monoms have different degrees");

return TNode<unsigned int, float>(monom.key, this->pData + monom.pData);

}

TNode<unsigned int, float> TNode<unsigned int, float>::operator-(const TNode<unsigned int, float>& monom)

{

if (this->key != monom.key)

throw M\_Exeption("monoms have different degrees");

return TNode<unsigned int, float>(monom.key, this->pData - monom.pData);

}

TNode<unsigned int, float>\* TNode<unsigned int, float>::operator-() const

{

TNode<unsigned int, float>\* tmp = new TNode<unsigned int, float>(\*this);

tmp->pData = -1 \* tmp->pData;

return tmp;

}

#endif

Файл TList.h

#ifndef TLIST\_H

#define TLIST\_H

#include<iostream>

#include "TNode.h"

#include"Exception.h"

using namespace std;

template<class TKey, class TData>

class TList

{

private:

TNode<TKey, TData>\* pFirst;

TNode<TKey, TData>\* pCurrent;

TNode<TKey, TData>\* pPrevious;

TNode<TKey, TData>\* pNext;

void Sort();

void BringingSimilar();

public:

TList();

TList(const TList&);

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

~TList();

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

void InsertBegin(TKey, TData);

void InsertBegin(TNode<TKey, TData>\*);

void InsertEnd(TKey, TData);

void InsertEnd(TNode<TKey, TData>\*);

void InsertAfter(TKey, TNode<TKey, TData>\*);

void InsertBefore(TKey, TNode<TKey, TData>\*);

void Remove(TKey);

bool IsEmpty()const;

bool IsEnded()const;

void Reset();

void Next();

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

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

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

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

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

{

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

while (tmp != nullptr)

{

out << "Key = " << tmp->key << endl;

tmp = tmp->pNext;

}

return out;

}

friend class Polinom;

};

template<class TKey, class TData>

TList<TKey, TData> ::TList()

{

pFirst = nullptr;

pCurrent = nullptr;

pPrevious = nullptr;

pNext = nullptr;

}

template<class TKey, class TData>

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

{

pPrevious = nullptr;

pNext = nullptr;

pCurrent = nullptr;

if (first == nullptr)

pFirst == nullptr;

else

{

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

pFirst = node;

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

TNode<TKey, TData>\* it = first->pNext;

while (it)

{

TNode<TKey, TData>\* tmp = new TNode<TKey, TData>(\*it);

prev->pNext = it;

prev = tmp;

it = it->pNext;

}

prev->pNext = nullptr;

pCurrent = pFirst;

pNext = pCurrent->pNext;

}

}

template<class TKey, class TData>

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

{

if (list.pFirst == nullptr)

{

pFirst = nullptr;

pCurrent = nullptr;

pPrevious = nullptr;

pNext = nullptr;

}

else

{

pFirst = new TNode<TKey, TData>(\*list.pFirst);

pPrevious = nullptr;

pCurrent = pFirst;

pNext = pFirst->pNext;

TNode<TKey, TData>\* tmp1(pFirst), \* tmp2(list.pFirst);

while (tmp2->pNext != nullptr)

{

tmp1->pNext = new TNode<TKey, TData>(\*(tmp2->pNext));

tmp2 = tmp2->pNext;

tmp1 = tmp1->pNext;

}

tmp1->pNext = nullptr;

}

}

template<class TKey, class TData>

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

{

while (pFirst != nullptr)

{

pNext = pFirst->pNext;

delete pFirst;

pFirst = pNext;

}

pCurrent = nullptr;

pPrevious = nullptr;

pNext = nullptr;

}

template<class TKey, class TData>

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

{

if (pFirst == nullptr)

return;

if (pFirst->pNext == nullptr)

return;

TNode<TKey, TData>\* a, \* b, \* p, \* h = NULL;

for (TNode<TKey, TData>\* i = pFirst; i != NULL; )

{

a = i;

i = i->pNext;

b = h;

for (p = NULL; (b != NULL) && (a->key < b->key); ) {

p = b;

b = b->pNext;

}

if (p == NULL) {

a->pNext = h;

h = a;

}

else {

a->pNext = b;

p->pNext = a;

}

}

if (h != NULL)

pFirst = h;

Reset();

}

template<class TKey, class TData>

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

{

Reset();

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

while (!IsEnded())

{

while (tmp->pNext->key != pCurrent->key)

{

if (pCurrent == tmp->pNext)

{

Next();

continue;

}

pCurrent->pData += tmp->pNext->pData;

TNode<TKey, TData>\* node = tmp->pNext;

tmp->pNext = tmp->pNext->pNext;

delete node;

Next();

}

}

}

template<class TKey, class TData>

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

{

if (pFirst == nullptr)

throw M\_Exeption("list is empty");

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

while ((tmp != nullptr) && (tmp->key != key))

tmp = tmp->pNext;

if (tmp == nullptr)

cout << "there is no such key";

return tmp;

}

template<class TKey, class TData>

void TList<TKey, TData> ::InsertBegin(TKey NewKey, TData NewData)

{

TNode<TKey, TData>\* tmp = new TNode<TKey, TData>(NewKey, NewData, pFirst);

if (pCurrent == pFirst)

pPrevious = tmp;

pFirst = tmp;

return;

}

template<class TKey, class TData>

void TList<TKey, TData> ::InsertBegin(TNode<TKey, TData>\* node)

{

InsertBegin(node->key, node->pData);

}

template<class TKey, class TData>

void TList<TKey, TData> ::InsertEnd(TKey NewKey, TData NewData)

{

if (pFirst == nullptr)

{

InsertBegin(NewKey, NewData);

Reset();

return;

}

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

while (tmp->pNext != 0)

tmp = tmp->pNext;

tmp->pNext = new TNode<TKey, TData>(NewKey, NewData);

if (pCurrent == tmp)

pNext = tmp->pNext;

tmp->pNext->pNext = nullptr;

return;

}

template<class TKey, class TData>

void TList<TKey, TData> ::InsertEnd(TNode<TKey, TData>\* node)

{

InsertEnd(node->key, node->pData);

}

template<class TKey, class TData>

void TList<TKey, TData> ::InsertAfter(TKey SearchKey, TNode<TKey, TData>\* node)

{

if (pFirst == nullptr)

throw M\_Exeption("list is empty");

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

while (tmp != nullptr && tmp->key != SearchKey)

tmp = tmp->pNext;

if (tmp == nullptr)

{

cout << "there is no such key" << endl;

return;

}

TNode<TKey, TData>\* NewNode = new TNode<TKey, TData>(node->key, node->pData, tmp->pNext);

if (pCurrent == tmp)

pNext = NewNode;

if (pCurrent == tmp->pNext)

pPrevious = NewNode;

tmp->pNext = NewNode;

return;

}

template<class TKey, class TData>

void TList<TKey, TData> ::InsertBefore(TKey SearchKey, TNode<TKey, TData>\* node)

{

if (pFirst == nullptr)

throw M\_Exeption("list is empty");

if (pFirst->key == SearchKey)

{

InsertBegin(node->key, node->pData);

return;

}

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

while (tmp != nullptr && tmp->pNext->key != SearchKey)

tmp = tmp->pNext;

if (tmp == nullptr)

{

cout << "there is no such key" << endl;

return;

}

TNode<TKey, TData>\* NewNode = new TNode<TKey, TData>(node->key, node->pData, tmp->pNext);

if (pCurrent == tmp->pNext)

pPrevious = NewNode;

if (pCurrent == tmp)

pNext = NewNode;

tmp->pNext = NewNode;

}

template<class TKey, class TData>

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

{

if (pFirst == nullptr)

throw M\_Exeption("list is empty");

if (pFirst->key == SearchKey)

{

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

pFirst = pFirst->pNext;

if (pCurrent == pFirst)

pPrevious = nullptr;

delete tmp;

Reset();

return;

}

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

while (tmp != nullptr && tmp->pNext->key != SearchKey)

tmp = tmp->pNext;

if (tmp == nullptr)

{

cout << "there is no such key for remove" << endl;

return;

}

TNode<TKey, TData>\* node = tmp->pNext;

tmp->pNext = tmp->pNext->pNext;

if (pCurrent == node)

Reset();

if (pCurrent == tmp)

pNext = tmp->pNext;

if (pCurrent == tmp->pNext)

pPrevious = tmp;

delete node;

}

template<class TKey, class TData>

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

{

return(pFirst == nullptr);

}

template<class TKey, class TData>

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

{

return(pCurrent == nullptr);

}

template<class TKey, class TData>

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

{

if (pFirst == nullptr)

return;

pCurrent = pFirst;

pPrevious = nullptr;

pNext = pCurrent->pNext;

}

template<class TKey, class TData>

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

{

if (IsEnded())

throw M\_Exeption("end of list");

pPrevious = pCurrent;

pCurrent = pNext;

if (pNext != nullptr)

pNext = pNext->pNext;

}

template<class TKey, class TData>

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

{

return pFirst;

}

template<class TKey, class TData>

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

{

return pCurrent;

}

template<class TKey, class TData>

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

{

return pNext;

}

template<class TKey, class TData>

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

{

return pPrevious;

}

#endif

Файл Polinom.h

#ifndef POLINOM\_H

#define POLINOM\_H

#include "TMonom.h"

#include <string>

#include <iostream>

#include "TList.h"

#include "Exception.h"

using namespace std;

class Polinom

{

private:

TList<unsigned int, float>\* monoms;

void SortMonoms();

void SimilarMonoms();

public:

Polinom();

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

Polinom(const string&);

Polinom(const Polinom& list);

~Polinom();

Polinom operator + (const Polinom& polynom)const;

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

Polinom operator - () const;

Polinom operator - (const Polinom& polynom);

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

Polinom operator \* (const Polinom& polynom)const;

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

bool operator ==(const Polinom& polynom)const;

const Polinom& operator= (const Polinom& polynom);

friend ostream& operator <<(std::ostream& out, const Polinom& pol);

};

Polinom::Polinom()

{

monoms = new TList<unsigned int, float>();

}

Polinom::Polinom(TList<unsigned int, float>\* list)

{

list->Reset();

while (!list->IsEnded())

{

if (list->GetCurrent()->key > 999 || list->GetCurrent()->key < 0)

throw M\_Exeption("invalid parameter");

list->Next();

}

monoms = new TList<unsigned int, float>(\*list);

SimilarMonoms();

SortMonoms();

}

Polinom::Polinom(const Polinom& list)

{

monoms = new TList<unsigned int, float>(\*list.monoms);

}

Polinom :: ~Polinom()

{

delete monoms;

}

void Polinom::SortMonoms()

{

if (monoms->GetFirst() == nullptr)

return;

if (monoms->GetFirst()->pNext == nullptr)

return;

TNode<unsigned int, float>\* a, \* b, \* p, \* h = nullptr;

for (TNode<unsigned int, float>\* i = monoms->GetFirst(); i != nullptr; ) {

a = i;

i = i->pNext;

b = h;

for (p = nullptr; (b != nullptr) && (a->key < b->key); ) {

p = b;

b = b->pNext;

}

if (p == nullptr) {

a->pNext = h;

h = a;

}

else {

a->pNext = b;

p->pNext = a;

}

}

if (h != nullptr)

monoms->pFirst = h;

monoms->Reset();

}

void Polinom::SimilarMonoms()

{

monoms->Reset();

while (!monoms->IsEnded())

{

TNode<unsigned int, float>\* tmp = monoms->GetNext();

TNode<unsigned int, float>\* prev\_tmp = monoms->GetCurrent();

while (tmp != nullptr)

{

if (monoms->GetCurrent()->key == tmp->key)

{

if (monoms->GetCurrent()->pData == -tmp->pData)

{

monoms->Remove(tmp->key);

monoms->Remove(tmp->key);

tmp = monoms->GetNext();

prev\_tmp = monoms->GetCurrent();

continue;

}

else

{

monoms->GetCurrent()->pData += tmp->pData;

TNode<unsigned int, float>\* node = tmp;

prev\_tmp->pNext = tmp->pNext;

if (monoms->GetNext() == node)

monoms->pNext = node->pNext;

delete node;

tmp = prev\_tmp->pNext;

continue;

}

}

tmp = tmp->pNext;

prev\_tmp = prev\_tmp->pNext;

}

monoms->Next();

}

monoms->Reset();

}

Polinom::Polinom(const string& polinom)

{

monoms = new TList<unsigned int, float>();

string str = polinom;

unsigned int i = 0;

float coeff = 1;

bool is\_x = false;

bool is\_y = false;

bool is\_z = false;

int deg\_x = 0;

int deg\_y = 0;

int deg\_z = 0;

while (i < str.length())

{

do

{

if (isdigit(str[i]) && !is\_x && !is\_y && !is\_z)

{

string StrCoeff;

while (isdigit(str[i]))

{

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

{

StrCoeff += str[i];

StrCoeff += str[i + 1];

i = i + 2;

}

else

{

StrCoeff += str[i];

i++;

}

}

coeff \*= (float)(atof(StrCoeff.c\_str()));

continue;

}

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

{

i++;

continue;

}

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

{

is\_x = true;

is\_y = false;

is\_z = false;

i++;

}

if (is\_x && (str[i] == 'y' || str[i] == 'z' || str[i] == ' ' ||

str[i] == '+' || str[i] == '-'))

{

deg\_x = 1;

is\_x = false;

continue;

}

if (is\_x && isdigit(str[i]))

{

if (isdigit(str[i + 1]))

throw M\_Exeption("degree > 9");

deg\_x = (int)str[i] - 48;

is\_x = false;

i++;

continue;

}

if (is\_x && (i == str.length()))

{

deg\_x = 1;

is\_x = false;

continue;

}

if (is\_x && (i + 1 == str.length()))

{

deg\_x = 1;

is\_x = false;

i++;

}

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

{

is\_x = false;

is\_y = true;

is\_z = false;

i++;

}

if (is\_y && (str[i] == 'x' || str[i] == 'z' || str[i] == ' ' ||

str[i] == '+' || str[i] == '-'))

{

deg\_y = 1;

is\_y = false;

continue;

}

if (is\_y && isdigit(str[i]))

{

if (isdigit(str[i + 1]))

throw M\_Exeption("degree > 9");

deg\_y = (int)str[i] - 48;

is\_y = false;

i++;

continue;

}

if (is\_y && i == str.length())

{

deg\_y = 1;

is\_y = false;

continue;

}

if (is\_y && i + 1 == str.length())

{

deg\_y = 1;

is\_y = false;

i++;

}

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

{

is\_x = false;

is\_y = false;

is\_z = true;

i++;

}

if (is\_z && (str[i] == 'x' || str[i] == 'y' || str[i] == ' ' ||

str[i] == '+' || str[i] == '-'))

{

deg\_z = 1;

is\_z = false;

continue;

}

if (is\_z && isdigit(str[i]))

{

if (isdigit(str[i + 1]))

throw M\_Exeption("degree > 9");

deg\_z = (int)str[i] - 48;

is\_z = false;

i++;

continue;

}

if (is\_z && i == str.length())

{

deg\_z = 1;

is\_z = false;

continue;

}

if (is\_z && i + 1 == str.length())

{

deg\_z = 1;

is\_z = false;

i++;

}

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

{

i++;

continue;

}

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

{

coeff \*= -1;

i++;

continue;

}

if (str[i] == ' ')

{

i++;

continue;

}

} while ((!is\_x || !is\_y || !is\_z) && !((str[i] == '+') ||

(str[i] == '-')) && (i != str.length()));

TNode<unsigned int, float>\* NewMon =

new TNode<unsigned int, float>(deg\_x, deg\_y, deg\_z, coeff);

monoms->InsertEnd(NewMon);

deg\_x = deg\_y = deg\_z = 0;

coeff = 1;

}

SimilarMonoms();

SortMonoms();

}

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

{

if (monoms->IsEmpty())

return polynom;

if (polynom.monoms->IsEmpty())

return \*this;

Polinom sum;

Polinom pol1(\*this);

Polinom pol2(polynom);

pol1.monoms->Reset();

pol2.monoms->Reset();

while (!pol1.monoms->IsEnded() && !pol2.monoms->IsEnded())

{

TNode<unsigned int, float>

m1(pol1.monoms->GetCurrent()->key, pol1.monoms->GetCurrent()->pData);

TNode<unsigned int, float>

m2(pol2.monoms->GetCurrent()->key, pol2.monoms->GetCurrent()->pData);

if (m1 > m2)

{

sum = sum + m1;

pol1.monoms->Next();

}

else if (m1 < m2)

{

sum = sum + m2;

pol2.monoms->Next();

}

else if (m1.key == m2.key)

{

if (m1.pData + m2.pData != 0)

{

TNode<unsigned int, float> SumMon = m1 + m2;

sum = sum + SumMon;

}

pol2.monoms->Next();

pol1.monoms->Next();

}

}

while (!pol2.monoms->IsEnded())

{

TNode<unsigned int, float>

m2(pol2.monoms->GetCurrent()->key, pol2.monoms->GetCurrent()->pData);

sum = sum + m2;

pol2.monoms->Next();

}

while (!pol1.monoms->IsEnded())

{

TNode<unsigned int, float>

m1(pol1.monoms->GetCurrent()->key, pol1.monoms->GetCurrent()->pData);

sum = sum + m1;

pol1.monoms->Next();

}

sum.monoms->Reset();

return sum;

}

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

{

return (Polinom(\*this) + (-polynom));

}

Polinom Polinom::operator-() const

{

Polinom tmp(\*this);

tmp.monoms->Reset();

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

{

tmp.monoms->pCurrent->pData = -tmp.monoms->pCurrent->pData;

tmp.monoms->Next();

}

tmp.monoms->Reset();

return tmp;

}

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

{

polynom.monoms->Reset();

monoms->Reset();

TNode<unsigned int, float>\* first = monoms->GetFirst();

TNode<unsigned int, float>\* second = polynom.monoms->GetFirst();

while (!polynom.monoms->IsEnded())

{

if (monoms->IsEnded() ||

polynom.monoms->GetCurrent() != monoms->GetCurrent())

{

monoms->Reset();

polynom.monoms->Reset();

return false;

}

monoms->Next();

polynom.monoms->Next();

}

if (!monoms->IsEnded())

return false;

monoms->Next();

polynom.monoms->Next();

return true;

}

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

{

if (\*this == polynom)

return \*this;

delete monoms;

monoms = new TList<unsigned int, float>(\*polynom.monoms);

return \*this;

}

Polinom Polinom::operator+(const TNode<unsigned int, float>& monom)

{

monoms->Reset();

TNode<unsigned int, float> pmonom(monom);

while (!monoms->IsEnded() && \*(monoms->GetCurrent()) > monom)

monoms->Next();

if (monoms->IsEnded())

{

monoms->InsertEnd(monom.key, monom.pData);

monoms->Reset();

return \*this;

}

if (\*(monoms->GetCurrent()) < monom)

{

monoms->InsertBefore(monoms->GetCurrent()->key, &pmonom);

}

if (monoms->GetCurrent()->key == monom.key)

{

if (monoms->GetCurrent()->pData + monom.pData != 0)

monoms->InsertAfter(monoms->GetCurrent()->key,

&(\*monoms->GetCurrent() + pmonom));

monoms->Remove(monoms->GetCurrent()->key);

monoms->Next();

}

monoms->Reset();

return \*this;

}

Polinom Polinom::operator-(const TNode<unsigned int, float>& monom)

{

return(Polinom(\*this) + TNode<unsigned int, float>(monom.key, -monom.pData));

}

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

{

if (polynom.monoms->IsEmpty())

throw M\_Exeption("polynomial is empty");

Polinom product;

Polinom pol(polynom);

Polinom copy(\*this);

pol.monoms->Reset();

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

{

TNode<unsigned int, float> tmp1(pol.monoms->GetCurrent()->key,

pol.monoms->GetCurrent()->pData);

product = product + copy \* tmp1;

pol.monoms->Next();

}

return product;

}

Polinom Polinom::operator\*(const TNode<unsigned int, float>& monom)const

{

Polinom res;

Polinom pol(\*this);

if (monom.pData == 0)

return res;

pol.monoms->Reset();

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

{

TNode<unsigned int, float> tmp(pol.monoms->GetCurrent()->key,

pol.monoms->GetCurrent()->pData);

tmp = tmp \* monom;

if (tmp.pData != 0)

res = res + tmp;

pol.monoms->Next();

}

return res;

}

ostream& operator <<(std::ostream& out, const Polinom& pol)

{

pol.monoms->Reset();

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

{

int deg\_x = pol.monoms->GetCurrent()->key / 100;

int deg\_y = (pol.monoms->GetCurrent()->key - deg\_x \* 100) / 10;

int deg\_z = pol.monoms->GetCurrent()->key - deg\_x \* 100 - deg\_y \* 10;

if (pol.monoms->GetCurrent()->pData == 0)

{

pol.monoms->Next();

continue;

}

else

{

if (pol.monoms->GetCurrent()->pData != 1 &&

pol.monoms->GetCurrent()->pData != -1)

{

if (pol.monoms->GetCurrent()->pData > 0)

out << "+" << pol.monoms->GetCurrent()->pData;

if (pol.monoms->GetCurrent()->pData < 0)

out << pol.monoms->GetCurrent()->pData;

}

if (pol.monoms->GetCurrent()->pData == -1)

out << "-";

if (pol.monoms->GetCurrent()->pData == 1 &&

pol.monoms->GetCurrent() != pol.monoms->GetFirst())

out << "+";

if (deg\_x > 0 && deg\_x != 1)

out << "x^" << deg\_x;

if (deg\_x == 1)

out << "x";

if (deg\_y > 0 && deg\_y != 1)

out << "y^" << deg\_y;

if (deg\_y == 1)

out << "y";

if (deg\_z > 0 && deg\_z != 1)

out << "z^" << deg\_z;

if (deg\_z == 1)

out << "z";

}

pol.monoms->Next();

}

return out;

}

#endif

Файл Exception.h

#pragma once

#include "TStack.h"

#include <string>

#include "Exсeption.h"

using namespace std;

class Postfix

{

private:

static int Prioritet(char v);

public:

static string PostfixForm(const string &v);

static void GetOperandsValues(const string& postfix,

char\*& operands, float\*& values, int& count);

static float Calculating(const string &v, char\* operands,

float\* values, int count);

};