Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования "Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского" (ННГУ)

Институт информационных технологий, математики и механики

Отчет по учебной практике

**Аналитические преобразования полиномов от нескольких переменных (списки)**

**Выполнил:**

Студент группы 381806-3

Лудина Д.А.

**Проверил:**

к.т.н., доцент каф. МОСТ ИИТММ

Кустикова В.Д.

Нижний Новгород

2020г.

**Содержание**

[Введение 3](#_Toc36055719)

[1 Постановка задач 4](#_Toc36055720)

[2 Руководство пользователя 5](#_Toc36055721)

[3 Руководство программиста 6](#_Toc36055722)

[3.1 Структура программы 6](#_Toc36055723)

[3.2 Описание структур данных 6](#_Toc36055724)

[3.2.1 Класс TNode 6](#_Toc36055725)

[3.2.2 Класс TList 6](#_Toc36055726)

[3.2.3 Класс монома 7](#_Toc36055727)

[3.2.4 Класс Polinom 8](#_Toc36055728)

[3.3 Описание алгоритмов 9](#_Toc36055729)

[3.3.1 Алгоритмы класса Polinom 9](#_Toc36055730)

[Заключение 11](#_Toc36055731)

[Литература 12](#_Toc36055732)

# Введение

С различными вычислениями человек сталкивается ежедневно. И сложно представить нашу жизнь без каких-либо подсчетов, потому что они встречаются на каждом шагу. Арифметические выражения являются неотъемлемой частью всех вычислительных программ, поэтому в составе языковых процессоров необходимо иметь алгоритмы, распознающие арифметические выражения с полиномами и вычисляющие их как можно быстрее. Полиномы являются хорошо изученной областью математики (алгебра полиномов), которая широко используется в приложениях.

Данная лабораторная работа направлена на практическое освоение методов организации структур хранения данных с помощью списков. В качестве области приложения выбираем тему вычисление арифметических выражений с полиномами.

# Постановка задач

В данной работе необходимо реализовать класс полинома на базе списка и обеспечить поддержку следующих операций:

* Сложение полиномов
* Вычитание полиномов
* Умножение полиномов

Также в классе должны быть следующие конструкторы:

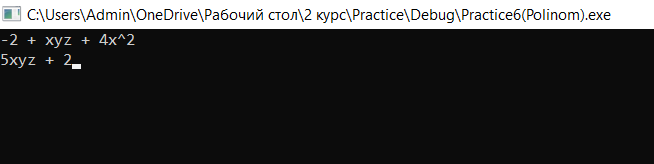
* Конструктор с параметром (строкой)
* Конструктор копирования
* Конструктор с параметром, принимающий на вход указатель на список и формирующий полином из этого списка

Вспомогательные операции: упорядочивание по степеням, приведение подобных.

А также основную программу, в которой запрашивается ввод полиномов и выполняется вычисление значения арифметических операций.

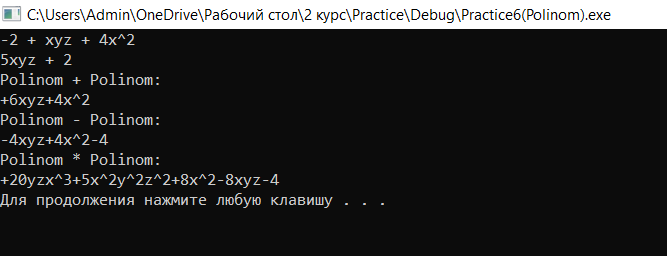
# Руководство пользователя

После запуска программы пользователю предлагается ввести два полинома. Вводим их и нажимаем Enter ().



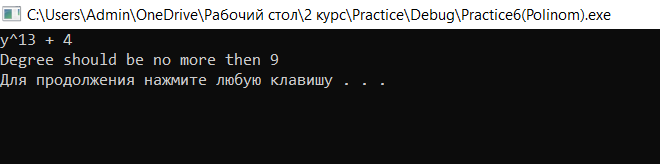
1. Ввод полиномов

Далее программа выведет значение арифметических операций (рис. 2).



1. Вывод значений арифметических операций

В случае ввода степени больше 9 (рис. 3).



1. Степень больше 9

# Руководство программиста

## Структура программы

Данная программа состоит из заголовочных файлов TNode.h, TMonom.h, TList.h, Polinom.h. А также из файлов ресурсов TMonom.cpp, Polinom.cpp и main.cpp.

В файле TNode.h осуществляется реализация шаблонного класса TNode. Он является звеном односвязного циклического списка. Этот файл подключен к файлу TList.h, в котором описан и реализован шаблонный класс TList. Он представляет собой односвязный циклический список. Файл TList.h подключен к файлу TMonom.h.

TMonom.h осуществляет описание класса который представляет собой слагаемое в полиноме. Реализация этого класса осуществляется в TMonom.cpp.

В файле Polinom.h осуществляется описание класса Polinom. Он является многочленом. Реализация этого класса осуществляется в файле Polinom.cpp, куда подключен файл Polinom.h. Также этот файл подключен к файлу maim.cpp

## Описание структур данных

### Класс TNode

Класс TNode – шаблонный.

Поля:

TKey key; идентификатор данных;

TData\* pData; указатель на данные;

TNode<TKey, TData>\* pNext; указатель на следующий элемент;

Помимо этого, в шаблонном классе TNode имеются следующие методы:

Node(); конструктор по умолчанию

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

TNode(TKey, TData\*); конструктор с параметрами

~TNode(); деструктор

### Класс TList

Класс TList – шаблонный.

Поля:

TNode<TKey, TData>\* pFirst; указатель на первый элемент списка

TNode<TKey, TData>\* pCurrent; указатель на текущий элемент списка

TNode<TKey, TData>\* pPrevious; указатель на предыдущий элемент списка

TNode<TKey, TData>\* pNext; указатель на следующий элемент списка

Методы:

TList(); конструктор по умолчанию

TList(const TList&); конструктор копирования

TList(TNode<TKey, TData>\* node); конструктор копирования

~TList(); деструктор

TNode<TKey, TData>\* Search(TKey); операция поиска по заданному ключу

void InsertBegin(TKey, TData); вставка в начало

void InsertEnd(TKey, TData); вставка в конец

void InsertAfter(TKey, TNode<TKey, TData>\*); вставка после элемента с заданным ключом

void InsertBefore(TKey, TNode<TKey, TData>\*); вставка до элемента с заданным ключом

void Remove(TKey); удаление элемента с заданным ключом

bool IsEmpty() const; проверка на пустоту

bool IsEnded() const; проверка на окончание списка

void Reset(); устанавливает текущий указатель в начало

void Next(); передвигает текущий указатель на следующий элемент

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

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

### Класс монома

Поля:

unsigned int degree; степень монома

double coeff; коэффициент перед мономом

TNode<unsigned int, double>\* pNext; указатель на следующий

Методы:

TNode(); конструктор по умолчанию

TNode(const TNode<unsigned int, double>&); конструктор копирования

TNode(unsigned int deg, double coef, TNode<unsigned int, double>\* node = nullptr); конструктор с параметрами

~TNode(); деструктор

bool operator<(const TNode&) const; перегрузка оператора сравнения

bool operator!=(const TNode& monom) const; перегрузка оператора сравнения

bool operator==(const TNode& monom) const; перегрузка оператора сравнения

TNode<unsigned int, double>\* operator\*(const TNode<unsigned int, double>& monom); перегрузка операции умножения двух мономов

### Класс Polinom

Поля:

TList<unsigned int, double>\* polinom; список из мономов

Методы:

Polinom(); конструктор по умолчанию

Polinom(const string& str); конструктор, преобразующий строку в полином

Polinom(const Polinom& tmp); конструктор копирования

Polinom(TList<unsigned int, double>\* list); конструктор с параметром

~Polinom(); деструктор

Polinom operator+(const Polinom&); перегрузка сложения полиномов

Polinom operator-(const Polinom&); перегрузка вычитания полиномов

Polinom operator\*(const Polinom&); перегрузка умножения полиномов

bool operator==(const Polinom&) const; перегрузка сравнения

const Polinom& operator=(const Polinom&); перегрузка оператора присваивания

Polinom operator-() const; перегрузка унарного минуса

Polinom operator\*(const TNode<unsigned int, double>& monom); перегрузка умножения монома на полином

friend ostream& operator<<(ostream& out, const Polinom& pol); оператор вывода

void OrganizeList(); упорядочивание по степеням (сортировка) полинома

void Simplification(); упрощение (приведение подобных)

## Описание алгоритмов

### Алгоритмы класса Polinom

1. Сложение полиномов:

Пока какой-либо из двух полиномов не закончился, выполняем следующие действия:

Сравниваем каждый моном.

1. Тот, который меньше, вставляем в конец созданного полинома, который и будет результатом сложения двух полиномов. Переходим на следующее слагаемое в том полиноме, моном которого оказался меньше.
2. Если мономы равны, то есть оказалось, что слагаемые из двух полиномов подобны, то складываем коэффициенты этих мономов и вставляем получившийся моном в конец результирующего полинома. Переходим на следующие слагаемые в каждом из двух полиномов.
3. Если какой-либо из двух полиномов не закончился, то вставляем в результирующий моном все слагаемые, которые остались в этом полиноме.
4. Вычитание полиномов:

Для вычитания из одного полинома другой сначала необходимо перегрузить унарный минус. Для этого проходим по полиному и пока он не закончится, умножаем каждый коэффициент на -1, а затем переходим на следующее слагаемое.

Операция вычитания возвращает сумму одного полинома и второго, к которому применена операция унарного минуса.

1. Умножение полиномов:

Для умножения двух полиномов сначала необходимо реализовать функцию умножения двух мономов в классе моном, затем функцию умножения монома на полином в классе Polinom, а потом требуемую операцию.

1. моном\*моном:

Создаем результирующий моном в котором коэффициент равен произведению коэффициентов двух мономов, а степень равна сумме их степеней.

1. моном\*полином:

Пока полином не закончился выполняем умножение коэффициентов монома и полинома и вставляем получившийся моном в конец результирующего полинома, затем переходим на следующее слагаемое в полиноме.

1. полином\*полином:

Берем моном из первого полинома и умножаем его на все мономы из второго, пока второй полином не закончился. Создаем моном с коэффициентом, который получается в результате умножения двух мономов из двух полиномов, и со степенью, которая получается в результате сложения степени одного монома и другого. Потом вставляем получившийся моном в конец результирующего полинома и переходим на следующий моном.

# Заключение

В ходе выполнения этой лабораторной работы мною были рассмотрены выполнения арифметических операций над полиномами, а именно сложение, вычитание и умножение двух полиномов. Я познакомилась с такой структурой хранения, как списки. На основе проделанной работы можно сделать вывод, что список позволяет управлять памятью наиболее эффективным образом и это значительно упрощает некоторые алгоритмы.

# Литература

1. Гергель, В.П. Методы программирования 1. Программа общего курса и описания лабораторных работ. Часть 1, — М.: Н.Новгород: ННГУ, 1999, — 151с
2. Кормен, Т., Лейзерсон, Ч., Ривест, Р. Алгоритмы: построение и анализ - М.: МЦНМО, 2000. — 960с.
3. Бьерн Страуструп. «Язык программирования С++»

Приложение

Файл TNode.h:

#pragma once

#include <iostream>

using namespace std;

template<class TKey, class TData>

class TNode

{

public:

TKey key;

TData\* pData;

TNode<TKey, TData>\* pNext;

TNode();

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

TNode(TKey, TData\*);

~TNode();

};

template<class TKey, class TData>

TNode<TKey, TData>::TNode()

{

key = 0;

pData = nullptr;

pNext = nullptr;

}

template<class TKey, class TData>

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

{

this->key = node.key;

this->pData = new TData(\*node.pData);

pNext = nullptr;

}

template<class TKey, class TData>

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

{

pNext = nullptr;

this->key = key;

this->pData = new TData(\*data.pData);

}

template<class TKey, class TData>

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

{

key = 0;

pNext = nullptr;

delete pData;

}

Файл TList.h:

#pragma once

#include "TNode.h"

#include <iostream>

using namespace std;

template<class TKey, class TData>

class TList

{

public:

TNode<TKey, TData>\* pFirst;

TNode<TKey, TData>\* pCurrent;

TNode<TKey, TData>\* pPrevious;

TNode<TKey, TData>\* pNext;

TList();

TList(const TList&);

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

~TList();

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

void InsertBegin(TKey, TData); //вставка в начало

void InsertEnd(TKey, TData); //вставка в конец

void InsertAfter(TKey, TNode<TKey, TData>\*); //вставка после элемента с заданым ключем

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

void Remove(TKey); //удаление элемента с данным ключем

bool IsEmpty() const; //проверка на пустоту

bool IsEnded() const; //дошли ли мы до конца списка

void Reset(); //устанавливает текущий указатель в начало

void Next(); //передвигает текущий указатель на следующий элемент

TNode<TKey, TData>\* GetFirst() const; //возвращает первый элемент

TNode<TKey, TData>\* GetCurrent() const; //возвращает текущий элемент

};

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 TList<TKey, TData>& list)

{

pCurrent = nullptr;

pPrevious = nullptr;

pNext = nullptr;

if (!list.pFirst)

{

pFirst = nullptr;

}

else

{

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

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

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

while (tmp->pNext != nullptr)

{

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

node = node->pNext;

tmp = tmp->pNext;

}

Reset();

}

}

template<class TKey, class TData>

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

{

pCurrent = nullptr;

pPrevious = nullptr;

pNext = nullptr;

if (node == nullptr)

pFirst == nullptr;

else

{

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

TNode<TKey, TData>\* node1 = pFirst;

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

while (tmp->pNext != nullptr)

{

node1->pNext = new TNode<TKey, TData>(\*(tmp->pNext));

node1 = node1->pNext;

tmp = tmp->pNext;

}

Reset();

}

}

template<class TKey, class TData>

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

{

pCurrent = nullptr;

pPrevious = nullptr;

pNext = nullptr;

while (!IsEnded())

{

pNext = pFirst->pNext;

delete pFirst;

pFirst = pNext;

}

}

template<class TKey, class TData>

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

{

if (pFirst == nullptr)

throw "list is empty";

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

while ((tmp != nullptr) && (tmp->degree != 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 key, TData data)

{

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

tmp->pNext = pFirst;

if (pCurrent == pFirst)

pPrevious = tmp;

pFirst = tmp;

}

template<class TKey, class TData>

void TList<TKey, TData>::InsertEnd(TKey key, TData data)

{

if (pFirst == nullptr)

{

InsertBegin(key, data);

Reset();

return;

}

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

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

TNode<TKey, TData>\* curr = pCurrent;

Reset();

while (!IsEnded())

Next();

pCurrent = new TNode<TKey, TData>(key, data);

pPrevious->pNext = pCurrent;

if (curr == pPrevious)

pNext = pCurrent;

else

pNext = next;

pCurrent = curr;

pPrevious = prev;

}

template<class TKey, class TData>

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

{

if (pFirst == nullptr)

throw"list is empty";

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

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

TNode<TKey, TData>\* curr = pCurrent;

Reset();

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

if (tmp == nullptr)

throw"Key isn't found";

while (pCurrent != tmp)

Next();

TNode<TKey, TData>\* node\_1 = new TNode<TKey, TData>(node->key, node->pData);

pCurrent->pNext = node\_1;

if (curr == pCurrent)

pNext = node\_1;

else

pNext = next;

if (curr == pNext)

pPrevious = node\_1;

else

pPrevious = prev;

pCurrent = curr;

}

template<class TKey, class TData>

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

{

if (pFirst == nullptr)

throw"list is empty";

if (pFirst->key == key)

{

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

return;

}

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

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

TNode<TKey, TData>\* curr = pCurrent;

Reset();

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

if (tmp == nullptr)

throw"Key isn't found";

while (pCurrent != tmp)

Next();

TNode<TKey, TData>\* node\_1 = new TNode<TKey, TData>(node->key, node->pData);

pPrevious->pNext = node\_1;

if (curr == pCurrent)

pPrevious = node\_1;

else

pPrevious = prev;

if (curr == pPrevious)

pNext = node\_1;

else

pNext = next;

pCurrent = curr;

}

template<class TKey, class TData>

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

{

if (pFirst == nullptr)

throw "list is empty";

if (pFirst->degree == key)

{

if (pCurrent == pFirst)

{

pCurrent = pNext;

if (pNext != nullptr)

pNext = pNext->pNext;

else

pNext = nullptr;

delete pFirst;

return;

}

if (pCurrent == pFirst->pNext)

{

pPrevious = nullptr;

delete pFirst;

pFirst = pCurrent;

return;

}

}

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

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

TNode<TKey, TData>\* curr = pCurrent;

Reset();

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

if (tmp == nullptr)

throw"Key isn't found 1";

while (pCurrent != tmp)

Next();

pPrevious->pNext = pNext;

if (curr == pCurrent)

{

pCurrent = pNext;

pNext = pCurrent->pNext;

delete tmp;

return;

}

if (curr == pNext)

{

pCurrent = pNext;

pNext = pCurrent->pNext;

delete tmp;

return;

}

if (curr == pPrevious)

{

pCurrent = pPrevious;

pPrevious = prev;

pNext = pCurrent->pNext;

delete tmp;

return;

}

pNext = next;

pCurrent = curr;

delete tmp;

return;

}

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()

{

pCurrent = pFirst;

pPrevious = nullptr;

if (pFirst != nullptr)

pNext = pCurrent->pNext;

else

pNext = nullptr;

}

template<class TKey, class TData>

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

{

if (IsEnded())

throw"End of list";

pPrevious = pCurrent;

pCurrent = pNext;

if (pNext != nullptr)

pNext = pNext->pNext;

else

pNext = nullptr;

}

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;

}

Файл TMonom.h:

#pragma once

#include "TList.h"

#include <iostream>

using namespace std;

template<>

class TNode<unsigned int, double>

{

public:

unsigned int degree;

double coeff;

TNode<unsigned int, double>\* pNext;

TNode();

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

TNode(unsigned int deg, double coef, TNode<unsigned int, double>\* node = nullptr);

~TNode();

bool operator<(const TNode&) const;

bool operator!=(const TNode& monom) const;

bool operator==(const TNode& monom) const;

TNode<unsigned int, double>\* operator\*(const TNode<unsigned int, double>& monom); //моном\*моном

};

Файл Polinom.h:

#pragma once

#include "TMonom.h"

#include <string>

using namespace std;

class Polinom

{

private:

TList<unsigned int, double>\* polinom;

void OrganizeList(); //упорядочивание по степеням(сортировка)

void Simplification(); //упрощение(приведение подобных)

public:

Polinom();

Polinom(const string& str);

Polinom(const Polinom& tmp); //контруктор копирования

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

~Polinom();

Polinom operator+(const Polinom&);

Polinom operator-(const Polinom&);

Polinom operator\*(const Polinom&);

bool operator==(const Polinom&) const;

const Polinom& operator=(const Polinom&);

Polinom operator-() const; //унарный минус

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

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

};

Файл TMonom.cpp:

#include "TMonom.h"

#include <iostream>

using namespace std;

TNode<unsigned int, double>::TNode()

{

degree = 0;

coeff = 0;

pNext = nullptr;

}

TNode<unsigned int, double>::TNode(const TNode<unsigned int, double>& n)

{

degree = n.degree;

coeff = n.coeff;

pNext = nullptr;

}

TNode<unsigned int, double>::TNode(unsigned int deg, double coef, TNode<unsigned int, double> \* node)

{

degree = deg;

coeff = coef;

pNext = node;

}

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

{

degree = 0;

coeff = 0;

pNext = nullptr;

}

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

{

return(this->degree < monom.degree);

}

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

{

return(this->degree != monom.degree);

}

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

{

return((this->degree == monom.degree) && (this->coeff == monom.coeff));

}

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

{

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

unsigned int deg\_x1 = monom.degree / 100;

unsigned int deg\_y1 = (monom.degree % 100) / 10;

unsigned int deg\_z1 = monom.degree % 10;

unsigned int deg\_x2 = this->degree / 100;

unsigned int deg\_y2 = (this->degree % 100) / 10;

unsigned int deg\_z2 = this->degree % 10;

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

throw "Degree shouldn't be > 9";

result->degree = this->degree + monom.degree;

result->coeff = this->coeff \* monom.coeff;

return result;

}

Файл Polinom.cpp:

#include "Polinom.h"

#include <string>

using namespace std;

Polinom::Polinom()

{

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

}

//isdigit() возвращает ненулевое значение, если аргумент ch является цифрой от 0 до 9, иначе 0

//atof() преобразует строку str в величину типа double

Polinom::Polinom(const string& str)

{

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

string s = str;

double coeff = 1;

int degree = 0;

int i = 0;

bool \_x = false;

bool \_y = false;

bool \_z = false;

while (i < s.length())

{

do

{

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

{

i++;

continue;

}

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

{

string c;

while (isdigit(s[i]))

{

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

{

c += s[i];

c += s[i + 1];

i = i + 2;

}

else

{

c += s[i];

i++;

}

}

coeff \*= (double)(atof(c.c\_str()));

continue;

}

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

{

i++;

\_x = true;

\_y = false;

\_z = false;

}

if ((\_x == true) && ((s[i] == 'y') || (s[i] == 'z') || (s[i] == ' ')))

{

\_x = false;

degree += 100;

}

if ((\_x == true) && (isdigit(s[i])))

{

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

throw "Degree should be no more then 9";

degree += ((int)s[i] - 48) \* 100;

\_x = false;

i++;

continue;

}

if (((i + 1) == s.length()) && (\_x == true))

{

\_x = false;

degree += 100;

i++;

}

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

{

i++;

\_x = false;

\_y = true;

\_z = false;

}

if ((\_y == true) && ((s[i] == 'x') || (s[i] == 'z') || (s[i] == ' ')))

{

\_y = false;

degree += 10;

}

if ((\_y == true) && (isdigit(s[i])))

{

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

throw "Degree should be no more then 9";

degree += ((int)s[i] - 48) \* 10;

\_y = false;

i++;

continue;

}

if (((i + 1) == s.length()) && (\_y == true))

{

\_y = false;

degree += 10;

i++;

}

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

{

i++;

\_x = false;

\_y = false;

\_z = true;

}

if (((i + 1) == s.length()) && (\_z == true))

{

\_z = false;

degree += 1;

i++;

continue;

}

if ((\_z == true) && !(isdigit(s[i])))

{

\_z = false;

degree += 1;

}

if ((\_z == true) && (isdigit(s[i])))

{

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

throw "Degree should be no more then 9";

degree += ((int)s[i] - 48);

\_z = false;

i++;

continue;

}

if ((s[i] == ' ') && (\_z == true))

{

\_z = false;

degree += 1;

}

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

{

i++;

continue;

}

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

{

i++;

coeff \*= (-1);

continue;

}

if (s[i] == ' ')

{

i++;

continue;

}

} while (!((s[i] == '+') || (s[i] == '-')) && (i != s.length()) && (!\_x || !\_y || !\_z));

polinom->InsertEnd(degree, coeff);

coeff = 1;

degree = 0;

}

Simplification();

OrganizeList();

}

Polinom::Polinom(const Polinom& tmp)

{

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

}

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

{

list->Reset();

while (!list->IsEnded())

{

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

throw"One of polynom's degree is wrong";

list->Next();

}

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

Simplification();

OrganizeList();

}

Polinom::~Polinom()

{

delete polinom;

}

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

{

if (polinom->IsEmpty())

return pol;

if (pol.polinom->IsEmpty())

return (\*this);

Polinom result;

pol.polinom->Reset();

polinom->Reset();

while ((!pol.polinom->IsEnded()) && (!polinom->IsEnded()))

{

if (polinom->GetCurrent()->degree < pol.polinom->GetCurrent()->degree)

{

result.polinom->InsertEnd(polinom->GetCurrent()->degree, polinom->GetCurrent()->coeff);

polinom->Next();

}

else if (polinom->GetCurrent()->degree > pol.polinom->GetCurrent()->degree)

{

result.polinom->InsertEnd(pol.polinom->GetCurrent()->degree, pol.polinom->GetCurrent()->coeff);

pol.polinom->Next();

}

else

{

if ((polinom->GetCurrent()->coeff + pol.polinom->GetCurrent()->coeff) != 0)

{

result.polinom->InsertEnd(polinom->GetCurrent()->degree, polinom->GetCurrent()->coeff + pol.polinom->GetCurrent()->coeff);

}

polinom->Next();

pol.polinom->Next();

}

}

while (!polinom->IsEnded())

{

result.polinom->InsertEnd(polinom->GetCurrent()->degree, polinom->GetCurrent()->coeff);

polinom->Next();

}

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

{

result.polinom->InsertEnd(pol.polinom->GetCurrent()->degree, pol.polinom->GetCurrent()->coeff);

pol.polinom->Next();

}

result.polinom->Reset();

result.OrganizeList();

result.Simplification();

return result;

}

Polinom Polinom::operator-() const

{

Polinom result(\*this);

while (!result.polinom->IsEnded())

{

result.polinom->GetCurrent()->coeff \*= -1;

result.polinom->Next();

}

result.polinom->Reset();

return result;

}

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

{

return \*this + (-pol);

}

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

{

if (monom.coeff == 0)

return polinom;

Polinom result;

TNode<unsigned int, double>\* mon = new TNode<unsigned int, double>(monom);

while (!polinom->IsEnded())

{

result.polinom->InsertEnd(polinom->GetCurrent()->degree, mon->coeff \* polinom->GetCurrent()->coeff);

polinom->Next();

}

Simplification();

OrganizeList();

return result;

}

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

{

Polinom result;

polinom->Reset();

while (!polinom->IsEnded())

{

pol.polinom->Reset();

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

{

TNode<unsigned int, double>\* res;

res = \*polinom->GetCurrent() \* \*pol.polinom->GetCurrent();

result.polinom->InsertEnd(res->degree, res->coeff);

pol.polinom->Next();

}

polinom->Next();

}

result.Simplification();

result.OrganizeList();

return result;

}

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

{

while (!pol.polinom->IsEnded() || !polinom->IsEnded())

{

if (polinom->GetCurrent() != pol.polinom->GetCurrent())

return false;

}

if (!pol.polinom->IsEnded() || !polinom->IsEnded())

return false;

else

return true;

}

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

{

if (\*this == pol)

return \*this;

if (!polinom->IsEnded())

{

delete polinom;

}

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

return \*this;

}

void Polinom::OrganizeList()

{

if ((polinom->GetFirst() == nullptr) || (polinom->GetFirst()->pNext == nullptr))

return;

polinom->Reset();

TNode<unsigned int, double>\* node = polinom->GetFirst();

TNode<unsigned int, double>\* node\_1 = polinom->GetFirst()->pNext;

TNode<unsigned int, double>\* prev\_node = polinom->GetFirst(), \*tmp;

while(node->pNext != nullptr)

{

while (node\_1 != nullptr)

{

if (node->degree < node\_1->degree)

{

if (prev\_node == node)

polinom->pFirst = node\_1;

else

prev\_node->pNext = node\_1;

node->pNext = node\_1->pNext;

node\_1->pNext = node;

tmp = node;

node = node\_1;

node\_1 = tmp;

}

prev\_node = node;

node = node->pNext;

node\_1 = node\_1->pNext;

}

}

}

void Polinom::Simplification()

{

polinom->Reset();

if (polinom->GetCurrent() == nullptr)

return;

while (!polinom->IsEnded())

{

TNode<unsigned int, double>\* tmp = polinom->GetFirst();

TNode<unsigned int, double>\* tmp\_1 = polinom->GetCurrent();

while (tmp->degree != tmp\_1->degree)

tmp = tmp->pNext;

if (tmp == tmp\_1)

{

polinom->Next();

continue;

}

polinom->GetCurrent()->coeff += tmp->coeff;

polinom->Remove(tmp->degree);

polinom->Next();

}

polinom->Reset();

}

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

{

unsigned int deg;

double coef;

pol.polinom->Reset();

if (pol.polinom->IsEnded())

{

out << "0";

return out;

}

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

{

deg = pol.polinom->GetCurrent()->degree;

coef = pol.polinom->GetCurrent()->coeff;

int deg\_x = deg / 100;

int deg\_y = (deg % 100) / 10;

int deg\_z = deg % 10;

if (coef == 0)

{

pol.polinom->Next();

continue;

}

else

{

if (coef != 1 && coef != -1)

{

if (coef > 0)

out << "+" << coef;

if (coef < 0)

out << coef;

}

if (coef == -1)

out << "-";

if (coef == 1)

out << "+";

if (deg\_x == 1)

out << "x";

if (deg\_y == 1)

out << "y";

if (deg\_z == 1)

out << "z";

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

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

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

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

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

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

}

pol.polinom->Next();

}

return out;

}

Файл main.cpp:

#include <iostream>

#include <string>

#include "Polinom.h"

using namespace std;

int main()

{

try

{

string s, s1;

getline(cin, s);

Polinom pol(s);

getline(cin, s1);

Polinom pol1(s1);

cout << "Polinom + Polinom:" << endl;

cout << pol + pol1 << endl;

cout << "Polinom - Polinom:" << endl;

cout << pol - pol1 << endl;

cout << "Polinom \* Polinom:" << endl;

cout << pol \* pol1 << endl;

}

catch (const char\* ex)

{

cout << ex << endl;

}

system("pause");

}