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

Отчёт по учебной практике

**Аналитические преобразования полиномов от нескольких переменных**

Выполнил: студент гр. 381806-3

Щекотилова Ю.А.

Проверил: к.т.н., доцент каф. МОСТ, ИИТММ

Кустикова В.Д.

Нижний Новгород

2019 г.

Содержание

[Введение 2](#_Toc27677886)

[1 Постановка задачи 3](#_Toc27677887)

[2 Руководство пользователя 4](#_Toc27677888)

[3 Руководство программиста 6](#_Toc27677889)

[3.1 Описание структуры программы 6](#_Toc27677890)

[3.2 Описание классов 6](#_Toc27677891)

[3.2.1 Класс TStack 6](#_Toc27677892)

[3.2.2 Класс Functions 7](#_Toc27677893)

[3.3 Описание алгоритмов 8](#_Toc27677894)

[3.3.1 Описание работы стека 8](#_Toc27677895)

[3.3.2 Описание перевода арифметического выражения в постфиксную форму 9](#_Toc27677896)

[3.3.3 Описание вычисления значения выражения по его постфиксной форме 10](#_Toc27677897)

[Заключение 12](#_Toc27677898)

[Литература 13](#_Toc27677899)

[Приложение 14](#_Toc27677900)

# Введение

Целью лабораторной работы является создание программы, способной выполнять арифметические операции (сложение, вычитание, умножение) с полиномами трех переменных со степенями от 0 до 9. Программа реализуется на языке С++. В качестве структуры данных используется список.

Разработанная программа может быть использована как самостоятельный продукт непосредственно для нахождения результата выполнения заданных пользователем арифметических операций с полиномами, так и в качестве составной части при решении более сложных задач.

# Постановка задачи

Разработать программу, которая позволяет пользователю задать полиномы, состоящие из мономов трех переменных x, y, z со степенями от 0 до 9 с вещественными коэффициентами, выполнить над ними заданные арифметические операции (сложения, вычитания, умножения двух полиномов) и вывести полученный результат вычислений.

# Руководство пользователя

Запускаем программу:

Перед пользователем появляется командная строка.

На данном этапе пользователь должен ввести корректный первый полином (Рис. 1):



Рис. 1. Основное окно программы. Ввод выражения.

Далее пользователь должен ввести второй полином. При правильном вводе программа выведет на экран сложение, вычитание и умножение двух полиномов (Рис. 2):

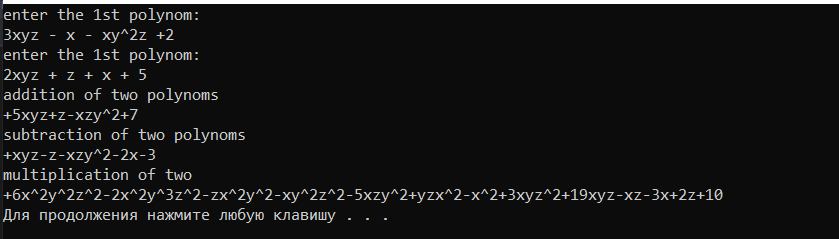


Рис. 2. Вывод арифметических выражений над полиномами

Если пользователь введет некорректные выражения, то программа выдаст сообщение об ошибке (Рис. 3):

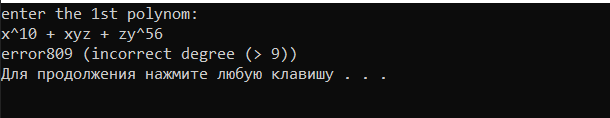


Рис. 3. Вывод ошибки при неправильно введеном выражении

# Руководство программиста

## Описание структуры программы

Программа состоит из следующих файлов:

* *Node.h,* который содержит шаблонный класс TNode*;*
* *List.h*, который содержит шаблонный класс TList ;
* *Monom.h*,который содержит класс Monom;
* *Polynom.h*, который содержит реализацию методов класса Functions.
* *main.cpp*

## Описание классов

### Класс TNode

**Объявление класса**

template<typename TKey, typename TData>

class TNode

{

public:

TKey key;

TData\* pData;

TNode<TKey, TData>\* pNext;

TNode();

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

TNode(TKey, TData\*);

~TNode();

};

**Описание полей**

TKey key

Назначение: переменная, которая идентифицирует данные.

TData\* pData

Назначение: указатель на данные.

TNode<TKey, TData>\* pNext

Назначение: переменная типа TNode, указывающая на следующий элемент.

### Класс TList

**Объявление класса**

template<typename TKey, typename TData>

class TList

{

public:

TNode<TKey, TData>\* pFirst;

TNode<TKey, TData>\* pCurr;

TNode<TKey, TData>\* pPrev;

TNode<TKey, TData>\* pNext;

TList();

TList(const TList&);

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

~TList();

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

void InsertBegin(TKey, TData);

void InsertEnd(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>\* getCurr() const;

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

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

};

**Описание полей**

TNode<TKey, TData>\* pFirst;

Назначение: указывает на первый элемент списка.

TNode<TKey, TData>\* pCurr;

Назначение: указывает на текущий элемент списка.

TNode<TKey, TData>\* pPrev;

Назначение: указывает на предыдущий элемент списка.

TNode<TKey, TData>\* pNext;

Назначение: указывает на следующий элемент списка.

**Описание методов**

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

Назначение: медот, предназначенный для поиска элемента по заданному ключу.

Входные данные: переменная типа TKey.

Выходные данные: указатель на переменную шаблонного типа TNode.

void InsertBegin(TKey, TData);

Назначение: вставка с заданным ключом элемента в начало.

Входные данные: переменная типа TKey и переменная типа TData.

void InsertEnd(TKey, TData);

Назначение: вставка заданным с заданным ключом ключом в конец.

Входные данные: переменная типа TKey и переменная типа TData.

void InsertAfter(TKey, TData);

Назначение: вставка после элемента с заданным ключом.

Входные данные: переменная типа TKey и переменная шаблонного типа TNode.

void InsertBefore(TKey, TData);

Назначение: вставка до элемента с заданным ключом.

Входные данные: переменная типа TKey и переменная шаблонного типа TNode.

void Remove(TKey);

Назначение: удаляет элемент с заданным ключом.

Входные данные: переменная типа TKey.

bool IsEmpty() const;

Назначение: проверяет на пустоту.

Выходные данные: значение true или false

bool IsEnded() const;

Назначение: проверяет список на заполненность.

Выходные данные: значение true или false

void Reset();

Назначение: устанавливает текущий указатель на элемент в начало.

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

Назначение: возвращает первый элемент.

Выходные данные: указатель на переменную шаблонного типа TNode.

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

Назначение: возвращает текущий элемент.

Выходные данные: указатель на переменную шаблонного типа TNode.

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

Назначение: возвращает предыдущий элемент.

Выходные данные: указатель на переменную шаблонного типа TNode.

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

Назначение: возвращает следующий элемент.

Выходные данные: указатель на переменную шаблонного типа TNode.

class TNode<unsigned int, float>

{

public:

unsigned int key;

float pData;

TNode<unsigned int, float>\* pNext;

TNode();

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

TNode(unsigned int, float, TNode<unsigned int, float>\* node = nullptr);

~TNode();

bool operator<(const TNode&) const;

bool operator!=(const TNode&) const;

bool operator==(const TNode&) const;

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

};

**Описание полей**

unsigned int key;

Назначение: задает идентификатор данных (степень монома).

float pData;

Назначение: задает коэффициет перед мономом.

TNode<unsigned int, float>\* pNext;

Назначение: указывает на следующий элемент.

**Описание методов**

bool operator<(const TNode&) const;

Назначение: перегрузка оператора сравнения.

bool operator!=(const TNode&) const;

Назначение: перегрузка оператора сравнения.

bool operator==(const TNode&) const;

Назначение: перегрузка оператора сравнения.

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

Назначение: перегрузка операции умножения монома на моном.

### Класс Polynom

class Polynom

{

private:

TList<unsigned int, float>\* monoms;

public:

Polynom();

Polynom(const string&);

Polynom(const Polynom&);

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

~Polynom();

void SortingDegs();

void SimularTerms();

Polynom operator+(const Polynom&);

Polynom operator-(const Polynom&);

Polynom operator\*(const Polynom&);

bool operator==(const Polynom&) const;

const Polynom& operator=(const Polynom&);

Polynom operator-() const;

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

friend ostream& operator<<(ostream& out, const Polynom&);

};

**Описание полей**

TList<unsigned int, float>\* monoms;

Назначение: список из мономов.

void SortingDegs();

Назначение: упорядочивает мономы по степеням.

void SimularTerms();

Назначение: приводит подобные.

Polynom operator+(const Polynom&);

Назначение: перегрузка операции сложения мономов.

Входные данные: ссылка на переменная типа Polynom.

Выходные данные: переменная типа Polynom.

Polynom operator-(const Polynom&);

Назначение: перегрузка операции вычитания мономов.

Входные данные: ссылка на переменная типа Polynom.

Выходные данные: переменная типа Polynom.

Polynom operator\*(const Polynom&);

Назначение: перегрузка операции умножения мономов.

Входные данные: ссылка на переменная типа Polynom.

Выходные данные: переменная типа Polynom.

bool operator==(const Polynom&) const;

Назначение: перегрузка операции сравнения.

Входные данные: константная ссылка на переменную типа Polynom.

Выходные данные: выводит значение true или false.

const Polynom& operator=(const Polynom&);

Назначение: перегрузка операции присваивания..

Polynom operator-() const;

Назначение: пегрузка операции унарного минуса.

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

Назначение: перегрузка операции умножения монома на полином.

Входные данные: переменную типа TNode.

Выходные данные: переменная типа Polynom.

friend ostream& operator<<(ostream& out, const Polynom&);

Назначение: перегрузка операции вывода полиномов.

## Описание алгоритмов

### Описание операции сложения

1. На вход поступает второй полином. Создается результирующий пустой полином. Далее алгоритм сложения аналогичен алгоритму слияния двух отсортированных списков:
2. Оба полинома просматриваются одновременно в цикле, пока один из них не кончился.
   1. Если степень текущего монома первого меньше, чем степень текущего монома второго, в результат вставляется текущий первого, иначе второго.
   2. В случае, если мономы одинаковые, а сумма их коэффициентов не равна нулю, вставляется моном с коэффициентом, равным сумме коэффициентов первого и второго монома.
   3. После выхода из цикла оставшиеся мономы вставляются в конец. Возвращается результат сложения

### Описание операции вычитания

1. Пегеружаем операцию унарного минуса.
   1. В цикле проходим по полиному и каждое слагаемое умножаем на (-1), а далее переходим ко второму слагаемому и проделываем то же самое.
2. В операции вычитания возвращаем сумму двух полиномов, так чтобы, ко второму было применена операция унарного минуса

### Описание операции умножения

1. Сначала необходимо реализовать операцию умножения монома на моном (реализована в файле Monom.h).
   1. Создаем моном (в него будем записывать результат), коэффициент которого равен произведению мономов, степень – сумме степеней
2. Далее реализуется умножение монома на полином
   1. Создаем цикл. Пока полином не закончился умножаем коэффициент монома и полинома.
   2. Получился полином, который мы вставляем в конец результирующего полинома, и так далее переходим на следующий моном в полиноме.
3. Умножение двух полиномов реализуется следующий образом:
   1. Первое слагаемое первого полинома умножаем на все мономы из второго полинома.
   2. Создаем такой моном, у которого коэффициент получается после умножения двух мономов, а степень – после сложения двух полиномов
   3. Получается моном, который вставляем в конец результирующего полинома и так далее.

# Заключение

Представленная программа позволяет пользователю задать полиномы, состоящие из мономов трех переменных x, y, z со степенями от 0 до 9 с вещественными коэффициентами, выполнить над ними заданные арифметические операции (сложения, вычитания, умножения двух полиномов) и вывести полученный результат вычислений. Таким образом, цель лабораторной работы достигнута.

# 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. — 960

# Приложение

Файл Node.h

#pragma once

#include <iostream>

using namespace std;

template<typename TKey, typename TData>

class TNode

{

public:

TKey key;

TData\* pData;

TNode<TKey, TData>\* pNext;

TNode();

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

TNode(TKey, TData\*);

~TNode();

};

template<typename TKey, typename TData>

TNode<TKey, TData>::TNode()

{

key = 0;

pData = nullptr;

pNext = nullptr;

}

template<typename TKey, typename TData>

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

{

this->key = node.key;

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

pNext = nullptr;

}

template<typename TKey, typename TData>

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

{

pNext = nullptr;

this->key = key;

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

}

template<typename TKey, typename TData>

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

{

key = 0;

pNext = nullptr;

delete pData;

}

Файл List.h

#pragma once

#include "Node.h"

using namespace std;

template<typename TKey, typename TData>

class TList

{

public:

TNode<TKey, TData>\* pFirst;

TNode<TKey, TData>\* pCurr;

TNode<TKey, TData>\* pPrev;

TNode<TKey, TData>\* pNext;

TList();

TList(const TList&);

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

~TList();

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

void InsertBegin(TKey, TData);

void InsertEnd(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>\* getCurr() const;

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

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

};

template<typename TKey, typename TData>

TList<TKey, TData>::TList()

{

pFirst = nullptr;

pPrev = nullptr;

pPrev = nullptr;

pNext = nullptr;

}

template<typename TKey, typename TData>

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

{

pCurr = nullptr;

pPrev = 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<typename TKey, typename TData>

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

{

pCurr = nullptr;

pPrev = 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<typename TKey, typename TData>

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

{

pCurr = nullptr;

pPrev = nullptr;

pNext = nullptr;

while (!IsEnded())

{

pNext = pFirst->pNext;

delete pFirst;

pFirst = pNext;

}

}

template<typename TKey, typename TData>

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

{

if (pFirst == nullptr) throw "error493 (list is empty)";

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

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

tmp = tmp->pNext;

if (tmp == nullptr) throw "error0980(key isn't found)";

return tmp;

}

template<typename TKey, typename TData>

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

{

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

tmp->pNext = pFirst;

if (pCurr == pFirst)

pPrev = tmp;

pFirst = tmp;

}

template<typename TKey, typename TData>

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

{

if (pFirst == nullptr)

{

InsertBegin(key, data);

Reset();

return;

}

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

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

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

Reset();

while (!IsEnded())

Next();

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

pPrev->pNext = pCurr;

if (curr == pPrev)

pNext = pCurr;

else

pNext = next;

pCurr = curr;

pPrev = prev;

}

template<typename TKey, typename TData>

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

{

if (pFirst == nullptr) throw "error493 (list is empty)";

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

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

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

Reset();

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

if (tmp == nullptr) throw "error0980 (key isn't found)";

while (pCurr != tmp) Next();

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

pCurr->pNext = node\_1;

if (curr == pCurr) pNext = node\_1;

else pNext = next;

if (curr == pNext) pPrev = node\_1;

else pPrev = prev;

pCurr = curr;

}

template<typename TKey, typename TData>

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

{

if (pFirst == nullptr) throw "error493 (list is empty)";

if (pFirst->key == key)

{

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

return;

}

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

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

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

Reset();

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

if (tmp == nullptr) throw "error0980 (key isn't found)";

while (pCurr != tmp) Next();

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

pPrev->pNext = node\_1;

if (curr == pCurr) pPrev = node\_1;

else pPrev = prev;

if (curr == pPrev) pNext = node\_1;

else pNext = next;

pCurr = curr;

}

template<typename TKey, typename TData>

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

{

if (pFirst == nullptr) throw "error0980 (key isn't found)";

if (pFirst-> key == key)

{

if (pCurr == pFirst)

{

pCurr = pNext;

if (pNext != nullptr)

pNext = pNext->pNext;

else

pNext = nullptr;

delete pFirst;

return;

}

if (pCurr == pFirst->pNext)

{

pPrev = nullptr;

delete pFirst;

pFirst = pCurr;

return;

}

}

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

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

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

Reset();

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

if (tmp == nullptr) throw "error0980 (key isn't found)";

while (pCurr != tmp) Next();

pPrev->pNext = pNext;

if (curr == pCurr)

{

pCurr = pNext;

pNext = pCurr->pNext;

delete tmp;

return;

}

if (curr == pNext)

{

pCurr = pNext;

pNext = pCurr->pNext;

delete tmp;

return;

}

if (curr == pPrev)

{

pCurr = pPrev;

pPrev = prev;

pNext = pCurr->pNext;

delete tmp;

return;

}

pNext = next;

pCurr = curr;

delete tmp;

return;

}

template<typename TKey, typename TData>

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

{

return (pFirst == nullptr);

}

template<typename TKey, typename TData>

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

{

return (pCurr == nullptr);

}

template<typename TKey, typename TData>

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

{

pCurr = pFirst;

pPrev = nullptr;

if (pFirst != nullptr)

pNext = pCurr->pNext;

else

pNext = nullptr;

}

template<typename TKey, typename TData>

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

{

if (IsEnded()) throw"error560 (list is ended)";

pPrev = pCurr;

pCurr = pNext;

if (pNext != nullptr)

pNext = pNext->pNext;

else

pNext = nullptr;

}

template<typename TKey, typename TData>

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

{

return pFirst;

}

template<typename TKey, typename TData>

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

{

return pCurr;

}

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 pPrev;

}

Файл Monom.h

#pragma once

#include "List.h"

using namespace std;

template<>

class TNode<unsigned int, float>

{

public:

unsigned int key;

float pData;

TNode<unsigned int, float>\* pNext;

TNode();

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

TNode(unsigned int, float, TNode<unsigned int, float>\* node = nullptr);

~TNode();

bool operator<(const TNode&) const;

bool operator!=(const TNode&) const;

bool operator==(const TNode&) const;

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

};

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

{

key = 0;

pData = 0;

pNext = nullptr;

}

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

{

key = n.key;

pData = n.pData;

pNext = nullptr;

}

TNode<unsigned int, float>::TNode(unsigned int deg, float coeff, TNode<unsigned int, float>\* node)

{

key = deg;

pData = coeff;

pNext = node;

}

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

{

key = 0;

pData = 0;

pNext = nullptr;

}

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

{

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

}

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

{

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

}

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

{

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

}

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

{

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

unsigned int d\_x1 = monom.key / 100;

unsigned int d\_y1 = (monom.key % 100) / 10;

unsigned int d\_z1 = monom.key % 10;

unsigned int d\_x2 = this->key / 100;

unsigned int d\_y2 = (this->key % 100) / 10;

unsigned int d\_z2 = this->key % 10;

if (d\_x1 + d\_x2 > 9 || d\_y1 + d\_y2 > 9 || d\_z1 + d\_z2 > 9) throw "error809 (incorrect key (> 9))";

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

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

return result;

}

Файл Polynom.h

#pragma once

#include "Monom.h"

#include <string>

using namespace std;

class Polynom

{

private:

TList<unsigned int, float>\* monoms;

public:

Polynom();

Polynom(const string&);

Polynom(const Polynom&);

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

~Polynom() { delete monoms; }

void SortingDegs();

void SimularTerms();

Polynom operator+(const Polynom&);

Polynom operator-(const Polynom&);

Polynom operator\*(const Polynom&);

bool operator==(const Polynom&) const;

const Polynom& operator=(const Polynom&);

Polynom operator-() const;

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

friend ostream& operator<<(ostream& out, const Polynom&);

};

Polynom::Polynom()

{

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

}

Polynom::Polynom(const string& str)

{

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

string s = str;

float pData = 1;

int key = 0, i = 0;

bool is\_x = false, is\_y = false, is\_z = false;

while (i < s.length())

{

do

{

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

{

i++;

continue;

}

if (isdigit(s[i]) && !is\_x && !is\_y && !is\_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++;

}

}

pData \*= (float)(atof(c.c\_str()));

continue;

}

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

{

i++;

is\_x = true;

is\_y = false;

is\_z = false;

}

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

{

is\_x = false;

key += 100;

}

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

{

if (isdigit(s[i + 1])) throw "error809 (incorrect key (> 9))";

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

is\_x = false;

i++;

continue;

}

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

{

is\_x = false;

key += 100;

i++;

}

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

{

i++;

is\_x = false;

is\_y = true;

is\_z = false;

}

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

{

is\_y = false;

key += 10;

}

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

{

if (isdigit(s[i + 1])) throw "error809 (incorrect key (> 9))";

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

is\_y = false;

i++;

continue;

}

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

{

is\_y = false;

key += 10;

i++;

}

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

{

i++;

is\_x = false;

is\_y = false;

is\_z = true;

}

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

{

is\_z = false;

key += 1;

i++;

continue;

}

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

{

is\_z = false;

key += 1;

}

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

{

if (isdigit(s[i + 1])) throw "error809 (incorrect key (> 9))";

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

is\_z = false;

i++;

continue;

}

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

{

is\_z = false;

key += 1;

}

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

{

i++;

continue;

}

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

{

i++;

pData \*= (-1);

continue;

}

if (s[i] == ' ')

{

i++;

continue;

}

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

monoms->InsertEnd(key, pData);

pData = 1;

key = 0;

}

SimularTerms();

SortingDegs();

}

Polynom::Polynom(const Polynom& tmp)

{

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

}

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

{

list->Reset();

while (!list->IsEnded())

{

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

throw "error8859 (some key is wrong)";

list->Next();

}

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

SimularTerms();

SortingDegs();

}

void Polynom::SortingDegs()

{

if (monoms->getFirst() == nullptr || monoms->getFirst()->pNext == nullptr) return;

monoms->Reset();

TNode<unsigned int, float>\* node = monoms->getFirst(), \*nnode = monoms->getFirst()->pNext;

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

while (node->pNext != nullptr)

{

while (nnode != nullptr)

{

if (node->key < nnode->key)

{

if (prev\_node == node)

monoms->pFirst = nnode;

else

prev\_node->pNext = nnode;

node->pNext = nnode->pNext;

nnode->pNext = node;

tmp = node;

node = nnode;

nnode = tmp;

}

prev\_node = node;

node = node->pNext;

nnode = nnode->pNext;

}

}

}

void Polynom::SimularTerms()

{

monoms->Reset();

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

return;

while (!monoms->IsEnded())

{

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

TNode<unsigned int, float>\* temp = monoms->getCurr();

while (tmp->key != temp->key)

tmp = tmp->pNext;

if (tmp == temp)

{

monoms->Next();

continue;

}

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

monoms->Remove(tmp->key);

monoms->Next();

}

monoms->Reset();

}

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

{

if (monoms->IsEmpty()) return polynom;

if (polynom.monoms->IsEmpty()) return (\*this);

Polynom result;

polynom.monoms->Reset();

monoms->Reset();

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

{

if (monoms->getCurr()->key < polynom.monoms->getCurr()->key)

{

result.monoms->InsertEnd(monoms->getCurr()->key, monoms->getCurr()->pData);

monoms->Next();

}

else if (monoms->getCurr()->key > polynom.monoms->getCurr()->key)

{

result.monoms->InsertEnd(polynom.monoms->getCurr()->key, polynom.monoms->getCurr()->pData);

polynom.monoms->Next();

}

else

{

if ((monoms->getCurr()->pData + polynom.monoms->getCurr()->pData) != 0)

{

result.monoms->InsertEnd(monoms->getCurr()->key, monoms->getCurr()->pData + polynom.monoms->getCurr()->pData);

}

monoms->Next();

polynom.monoms->Next();

}

}

while (!monoms->IsEnded())

{

result.monoms->InsertEnd(monoms->getCurr()->key, monoms->getCurr()->pData);

monoms->Next();

}

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

{

result.monoms->InsertEnd(polynom.monoms->getCurr()->key, polynom.monoms->getCurr()->pData);

polynom.monoms->Next();

}

result.monoms->Reset();

result.SortingDegs();

result.SimularTerms();

return result;

}

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

{

return \*this + (-polynom);

}

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

{

Polynom result;

monoms->Reset();

while (!monoms->IsEnded())

{

polynom.monoms->Reset();

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

{

TNode<unsigned int, float>\* res;

res = \*monoms->getCurr() \* (\*polynom.monoms->getCurr());

result.monoms->InsertEnd(res->key, res->pData);

polynom.monoms->Next();

}

monoms->Next();

}

result.SimularTerms();

result.SortingDegs();

return result;

}

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

{

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

{

if (monoms->getCurr() != polynom.monoms->getCurr())

return false;

}

if (!polynom.monoms->IsEnded() || !monoms->IsEnded())

return false;

else

return true;

}

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

{

if (\*this == polynom)

return \*this;

if (!monoms->IsEnded())

{

delete monoms;

}

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

return \*this;

}

Polynom Polynom::operator-() const

{

Polynom result(\*this);

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

{

result.monoms->getCurr()->pData \*= -1;

result.monoms->Next();

}

result.monoms->Reset();

return result;

}

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

{

if (monom.pData == 0)

return monoms;

Polynom result;

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

while (!monoms->IsEnded())

{

result.monoms->InsertEnd(monoms->getCurr()->key, mon->pData \* monoms->getCurr()->pData);

monoms->Next();

}

SimularTerms();

SortingDegs();

return result;

}

ostream& operator<<(ostream& out, const Polynom& polynom)

{

unsigned int d;

float coeff;

polynom.monoms->Reset();

if (polynom.monoms->IsEnded())

{

out << "0";

return out;

}

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

{

d = polynom.monoms->getCurr()->key;

coeff = polynom.monoms->getCurr()->pData;

int d\_x = d / 100;

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

int d\_z = d % 10;

if (coeff == 0)

{

polynom.monoms->Next();

continue;

}

else

{

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

{

if (coeff > 0)

out << "+" << coeff;

if (coeff < 0)

out << coeff;

}

if (coeff == -1)

out << "-";

if (coeff == 1)

out << "+";

if (d\_x == 1)

out << "x";

if (d\_y == 1)

out << "y";

if (d\_z == 1)

out << "z";

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

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

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

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

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

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

}

polynom.monoms->Next();

}

return out;

}

Файл main.cpp

#include "Polynom.h"

using namespace std;

int main()

{

try

{

string s, ss;

cout << "enter the 1st polynom:" << endl;

getline(cin, s);

Polynom polynom(s);

cout << "enter the 2nd polynom:" << endl;

getline(cin, ss);

Polynom pol(ss);

cout << "addition of two polynoms" << endl << endl;

cout << polynom + pol << endl;

cout << "subtraction of two polynoms" << endl << endl;

cout << polynom - pol << endl;

cout << "multiplication of two polynoms" << endl << endl;

cout << polynom \* pol << endl << endl;

}

catch (const char\* a) { cout << a << endl; }

system ("pause");

}