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

**Полином от нескольких переменных**

Выполнил:

студент гр. 381806 – 1

Соловьев А.Ю.

Проверил:

к.т.н., доцент каф. МОСТ, ИИТММ

Кустикова В.Д.

Нижний Новгород

2020 г.

Содержание

[Введение 3](#_Toc35790880)

[Постановка задачи 4](#_Toc35790881)

[Руководство пользователя 5](#_Toc35790882)

[Руководство программиста 6](#_Toc35790883)

[Описание структуры программы 6](#_Toc35790884)

[Описание структур данных 6](#_Toc35790885)

[Класс TNode 6](#_Toc35790886)

[Класс TList 7](#_Toc35790887)

[Класс Monom 10](#_Toc35790888)

[Класс Polynom 12](#_Toc35790889)

[Описание алгоритмов 16](#_Toc35790890)

[Список 16](#_Toc35790891)

[Полином 18](#_Toc35790892)

[Заключение 21](#_Toc35790893)

[Литература 22](#_Toc35790894)

[Приложения 23](#_Toc35790895)

[Приложение 1. Основная функция 23](#_Toc35790896)

[Приложение 2. Класс TNode 23](#_Toc35790897)

[Приложение 3. Класс TList 24](#_Toc35790898)

[Приложение 4. Класс Monom 29](#_Toc35790899)

[Приложение 5. Класс Polynom 32](#_Toc35790900)

# Введение

Целью данной лабораторной работы является практическое освоение методов организации структур хранения данных с помощью списков. В ходе выполнения лабораторной работы я разработал общую форму представления списков, для работы с полиномами.

# Постановка задачи

Задача: разработать алгоритм и составить программу для работы с полиномами от нескольких переменных.

Входные данные: 2 полинома от нескольких переменных в виде строк с форматированием вида -3\*x^3\*y^2\*z-4\*x^4\*y^4\*z^4+3\*x^2\*y\*z^3.

Выходные данные: полином от нескольких переменных в виде строки с форматированием вида -3\*x^3\*y^2\*z-4\*x^4\*y^4\*z^4+3\*x^2\*y\*z^3.

# Руководство пользователя

После запуска программы откроется консольный интерфейс с предложением ввести первый полином.(рекомендуется вводить полином -2+5x^2y+z-yz+x^2yz)

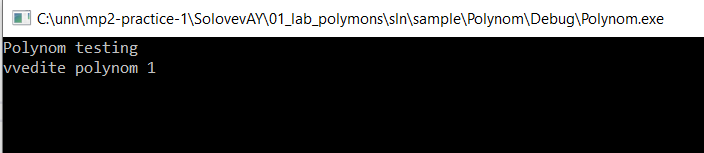


Рис. 1 Стартовый экран

После введения первого полинома нажмите клавишу Enter. Если введенный вами полином корректен, то программа предложит ввести второй полином. (рекомендуется вводить полином 2 + y - 2z + x ^ 2y)

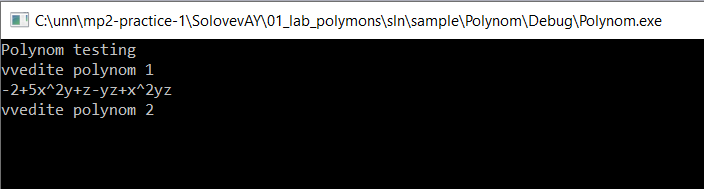


Рис. 2 Программа ожидает ввод второго полинома

После введения второго полинома нажмите клавишу Enter. Если введенный вами полином корректен, то программа выполнит сложение, вычитание и умножение введенных полиномов и выведет результат на экран.

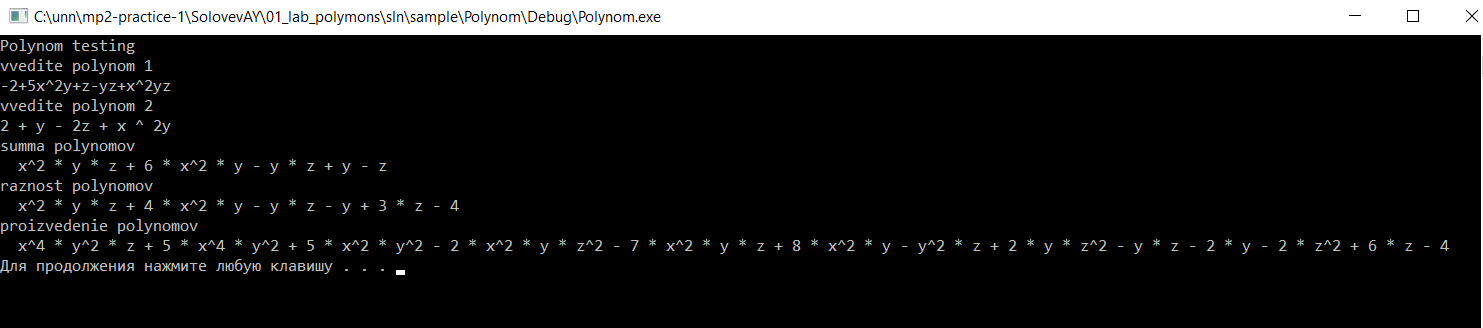


Рис. 3 Выведен результат

Если вы введете некорректный полином, то программа выведет сообщение об ошибке и завершит работу.

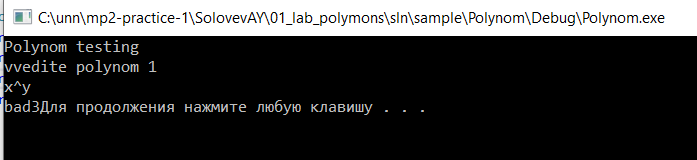


Рис. 4 Введен некорректный полином

# Руководство программиста

## Описание структуры программы

TNode.h – модуль с шаблонным классом, содержащий описание и реализацию функций звена списка.

TList.h – модуль с шаблонным классом, содержащий описание и реализацию функций списка.

Monom.h – модуль, содержащий описание специализированного класс TNode<double, unsigned int>, который выполняет роль монома.

Polynom.h – модуль, содержащий описание полинома.

## Описание структур данных

### Класс TNode

TNode – класс, реализующий звено списка.

**Объявление:**

template <typename TKey, typename TData>

struct TNode

{

TKey key;

TData data;

TNode\* pNext;

TNode();

TNode(TKey \_key, TData \_data);

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

~TNode();

const TNode<TKey, TData>& operator=(const TNode<TKey, TData>& \_tnode);

};

Поля класса:

data – данные;

key – ключ;

pNext – указатель на следующее звено.

**Описание методов класса:**

TNode(TKey \_key, TData \_data);

Назначение: конструктор, инициализация объекта

Входные параметры: элементы типа TKey, TData и указатель на звено

Возвращаемое значение: нет

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

Назначение: конструктор копирования, создание копии текущего стека

Входные параметры: константная ссылка на копируемый объект

Возвращаемое значение: нет

~TNode();

Назначение: деструктор, высвобождение памяти и удаление объекта класс

Входные параметры: нет

Возвращаемое значение: нет

const TNode<TKey, TData>& operator=(const TNode<TKey, TData>& \_tnode);

Назначение: оператор равенства двух звеньев

Входные параметры: константная ссылка на объект типа TNode

Возвращаемое значение: логическая переменная

### Класс TList

TList– класс, реализующий список

**Объявление:**

template <typename TKey, typename TData>

class TList

{

protected:

TNode<TKey, TData>\* pFirst;

TNode<TKey, TData>\* pNext;

TNode<TKey, TData>\* pPrevious;

TNode<TKey, TData>\* pCurrent;

public:

TList();

TList(const TList<TKey, TData>& \_tlist);

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

~TList();

void InsertBegin(TKey \_newKey, TData \_data);

void InsertEnd(TKey \_newKey, TData \_data);

void InsertBefore(TKey \_key, TKey \_newKey, TData \_data);

void InsertAfter(TKey \_key, TKey \_newKey, TData \_data);

void InsertBegin(const TNode<TKey, TData>& \_tnode);

void InsertEnd(const TNode<TKey, TData>& \_tnode);

void InsertBefore(TKey \_key, const TNode<TKey, TData>& \_tnode);

void InsertAfter(TKey \_key, const TNode<TKey, TData>& \_tnode);

void Remove(TKey \_key);

void Reset();

void Next();

bool IsEnded() const;

bool IsEmpty() const;

TKey getCurrentNodeKey() const;

TData getCurrentNodeData() const;

};

**Поля класса:**

pFirst – указатель на первый элемент списка;

pPrevious – указатель на предыдущий элемент списка.

pCurrent – указатель на текущий элемент списка;

pNext – указатель на следующий элемент списка;

**Описание методов класса:**

TList()

Назначение: конструктор по умолчанию.

Входные параметры: нет.

Возвращаемое значение: нет.

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

Назначение: конструктор копирования

Входные параметры: указатель на первое звено списка

Возвращаемое значение: нет

TList(const TList<TKey, TData>& \_tlist);

Назначение: конструктор копирования

Входные параметры: константная ссылка на копируемый список

Возвращаемое значение: нет

~TList()

Назначение: деструктор, высвобождение памяти и удаление объекта класс

Входные параметры: нет

Возвращаемое значение: нет

bool IsEnded() const

Назначение: проверка на окончание списка

Входные параметры: нет

Возвращаемое значение: логическая переменная

void Next()

Назначение: переход к следующему элементу

Входные параметры: нет

Возвращаемое значение: нет

void Reset()

Назначение: переход к первому элементу списка

Входные параметры: нет

Возвращаемое значение: нет

void InsertEnd(TKey \_newKey, TData \_data);

Назначение: вставка элемента в конец списка

Входные параметры: ключ и данные вставляемого элемента

Возвращаемое значение: нет

void InsertBegin(TKey \_newKey, TData \_data);

Назначение: вставка элемента в начало списка

Входные параметры: ключ и данные вставляемого элемента

Возвращаемое значение: нет

void InsertAfter(TKey, TData, TKey);

Назначение: вставка элемента в список после заданного элемента

Входные параметры: ключ и данные вставляемого элемента, ключ элемента после которого необходимо выполнить вставку

Возвращаемое значение: нет

void InsertBefore(TKey, TData, TKey);

Назначение: вставка элемента в список до заданного элемента

Входные параметры: ключ и данные вставляемого элемента, ключ элемента до которого необходимо выполнить вставку

Возвращаемое значение: нет

void Remove(TKey)

Назначение: удаление элемента с заданным ключом

Входные параметры: ключ элемента, который необходимо удалить

Возвращаемое значение: нет

TKey getCurrentNodeKey() const;

Назначение: возвращает ключ текущего элемента

Входные параметры: нет

Возвращаемое значение: указатель на элемент списка

### Класс Monom

Monom – класс реализующий моном.

**Объявление:**

#define Monom TNode<UINT, double>

template<>

struct Monom

{

UINT key;//degree

double koef;//koef

Monom\* pNext;

TNode();

TNode(UINT \_key, double \_koef);

TNode(double \_koef);

TNode(const Monom& \_monom);

~TNode();

bool operator==(const Monom& \_monom) const;

bool operator!=(const Monom& \_monom) const;

const Monom& operator=(const Monom& \_monom);

Monom operator-() const;

Monom operator+(const Monom& \_monom);

Monom operator-(const Monom& \_monom);

Monom operator\*(const Monom& \_monom);

bool operator>(const Monom& \_monom) const;

bool operator<(const Monom& \_monom) const;

char znakmonoma();

friend istream& operator>>(istream& in, Monom& \_monom);

friend ostream& operator<<(ostream& out, const Monom& \_monom);

static Monom convert(const string& \_monom);

};

Поля класса:

key – степень

koef– коэффициент

pNext – указатель на следующий моном

**Описание методов класса:**

TNode(const Monom& \_monom);

Назначение: конструктор копирования, создание копии текущего стека

Входные параметры: константная ссылка на копируемый объект

Возвращаемое значение: нет

TNode(UINT \_key, double \_koef);

Назначение: конструктор, инициализация объекта

Входные параметры: степень, коэффициент и указатель на следующий моном

Возвращаемое значение: нет

~TNode();

Назначение: деструктор, высвобождение памяти и удаление объекта класс

Входные параметры: нет

Возвращаемое значение: нет

Monom operator+(const Monom& \_monom);

Назначение: оператор сложения двух мономов

Входные параметры: константная ссылка на моном

Возвращаемое значение: результат сложения

Monom operator-(const Monom& \_monom);

Назначение: оператор вычитания двух мономов

Входные параметры: константная ссылка на моном

Возвращаемое значение: результат вычитания

Monom operator\*(const Monom& \_monom);

Назначение: оператор умножения двух мономов

Входные параметры: константная ссылка на моном

Возвращаемое значение: результат умножения

Monom operator-() const;

Назначение: унарный минус

Входные параметры: нет

Возвращаемое значение: моном с противоположным знаком

bool operator>(const Monom& \_monom) const;

Назначение: оператор больше

Входные параметры: константная ссылка на моном

Возвращаемое значение: логическая переменная

bool operator<(const Monom& \_monom) const;

Назначение: оператор меньше

Входные параметры: константная ссылка на моном

Возвращаемое значение: логическая переменная

bool operator==(const Monom& \_monom) const;

Назначение: оператор равенства

Входные параметры: константная ссылка на моном

Возвращаемое значение: логическая переменная

bool operator!=(const Monom& \_monom) const;

Назначение: оператор неравенства

Входные параметры: константная ссылка на моном

Возвращаемое значение: логическая переменная

### Класс Polynom

Polynom – класс, реализующий полином.

**Объявление:**

class Polynom

{

private:

static Polynom convert(const string& \_expression);

protected:

TList<UINT, double> \*monoms;

public:

Polynom();

Polynom(const string& \_expression);

Polynom(const TList<UINT, double>& \_list);

Polynom(const Monom& \_monom);

Polynom(const Polynom& \_polynom);

~Polynom();

Polynom& operator=(const Polynom& \_polynom);

bool operator==(const Polynom& \_polynom);

Polynom operator-() const;

Polynom operator+(const Polynom& \_polynom) const;

Polynom operator+(const Monom& \_monom) const;

Polynom operator-(const Polynom& \_polynom) const;

Polynom operator-(const Monom& \_monom) const;

Polynom operator\*(const Polynom& \_polynom) const;

Polynom operator\*(const Monom& \_monom) const;

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

friend istream& operator>>(istream& in, Polynom& \_polynom);

};

**Поля класса:**

monoms – список мономов

**Описание методов класса:**

Polynom();

Назначение: конструктор по умолчанию

Входные параметры: нет

Возвращаемое значение: нет

Polynom(const Polynom& \_polynom);

Назначение: конструктор копирования, создание копии текущего стека

Входные параметры: константная ссылка на копируемый объект

Возвращаемое значение: нет

Polynom(const string& \_expression);

Назначение: конструктор, инициализация объекта

Входные параметры: константная ссылка на строку

Возвращаемое значение: нет

Polynom(const TList<UINT, double>& \_list);

Назначение: конструктор, инициализация объекта

Входные параметры: константная ссылка на список

Возвращаемое значение: нет

~Polynom()

Назначение: деструктор, высвобождение памяти и удаление объекта класс

Входные параметры: нет

Возвращаемое значение: нет

Polynom operator+(const Monom& \_monom) const;

Назначение: оператор сложения полинома и монома

Входные параметры: константная ссылка на моном

Возвращаемое значение: результат сложения

Polynom operator-(const Monom& \_monom) const;

Назначение: оператор вычитания монома из полинома

Входные параметры: константная ссылка на моном

Возвращаемое значение: результат вычитания

Polynom operator\*(const Monom& \_monom) const;

Назначение: оператор умножения полинома и монома

Входные параметры: константная ссылка на моном

Возвращаемое значение: результат умножения

Polynom operator+(const Polynom& \_polynom) const;

Назначение: оператор сложения полиномов

Входные параметры: константная ссылка на полином

Возвращаемое значение: результат сложения

Polynom operator-(const Polynom& \_polynom) const;

Назначение: оператор вычитания полиномов

Входные параметры: константная ссылка на полином

Возвращаемое значение: результат вычитания

Polynom operator\*(const Polynom& \_polynom) const;

Назначение: оператор умножения полиномов

Входные параметры: константная ссылка на полином

Возвращаемое значение: результат умножения

Polynom& operator=(const Polynom& \_polynom);

Назначение: оператор присваивания

Входные параметры: константная ссылка на полином

Возвращаемое значение: ссылка на полином

bool operator==(const Polynom& \_polynom);

Назначение: оператор равенства

Входные параметры: константная ссылка на полином

Возвращаемое значение: логическая переменная

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

Назначение: вывод полинома

Входные параметры: поток вывода, константная ссылка на объект типа Polynom.

Возвращаемое значение: поток вывода

friend istream& operator>>(istream& in, Polynom& \_polynom);

Назначение: ввод полинома

Входные параметры: поток ввода, константная ссылка на объект тип Polynom

Возвращаемое значение: поток ввода

## Описание алгоритмов

### Список

Для работы со списками предлагается реализовать следующие операции:

методы проверка на пустоту и проверки на полноту списка;

методы навигации по списку (итератор);

методы вставки в начало, в конец, после звена с заданным ключи и перед звеном с заданным ключом;

методы удаления звена с заданным ключом и поиск звена с заданным ключом.

#### Вставка в начало

1. Создать новое звено списка temp.
2. Установить указатель на следующее звено у temp равным pFirst.
3. Поменять указатель начала списка на temp.

#### Вставка в конец

1. Установить указатель temp на последнее звено списка.
2. Создать новое звено node.
3. Установить указатель на следующее звено у temp равным node.
4. Установить указатель на следующее звено у node равным NULL.

#### Вставка после заданного элемента

1. Найти звено с заданным ключом и записать его в temp.
2. Создать новое звено node.
3. Установить указатель на следующее звено у node равным следующим у temp.
4. Установить указатель на следующее звено у temp равным node.

#### Вставка до заданного элемента

1. Найти звено, которое стоит перед звеном с заданным ключом и записать его в temp.
2. Создать новое звено node.
3. Установить указатель на следующее звено у node равным следующим у temp.
4. Установить указатель на следующее звено у temp равным node.

#### Удаление звена

1. Записать указатель на звено, стоящее перед удаляемым, в temp.
2. Записать указатель на удаляемое звено в node.
3. Записать указатель на звено, стоящее после удаляемого, в next.
4. Установить указатель на следующее звено у temp равным next.
5. Удалить node.

### Полином

Для организации быстрого доступа может быть использовано упорядоченное хранение мономов. Для задания порядка следования можно принять лексикографическое упорядочивание по степеням переменных, при котором мономы упорядочиваются по степеням первой переменной, потом по второй переменной, и только затем по третьей переменной. В общем виде это правило можно записать как соотношение: моном предшествует моному тогда и только тогда, если

(A1 > A2) ˅ (A1 = A2) & (B1 > B2) ˅ (A1 = A2) & (B1 = B2) & (C1 > C2).

Проверка лексикографического порядка занимает сравнительно много времени. Ее можно существенно упростить при помощи свертки, образуемой с использованием позиционной системы счисления: для монома со степенями (A, B, C) ставится в соответствие величина

ABC = A \* 100 + B \* 10 + C.

# Заключение

В ходе выполнения лабораторной работы, я изучил структуру данных, принципы работы и возможные операции над списками. Рассмотрел и реализовал полином.

# Литература

# Приложения

## Приложение 1. Основная функция

int main()

{

char a;

Monom monom;

Monom testmono1;

Polynom testpoly1;

Monom testmono2;

Polynom testpoly2;

Polynom result;

int choose = 0;

int choosemonom = 0;

int choosepolynom = 0;

try

{

cout << "Polynom testing" << endl;

cout << "vvedite polynom 1" << endl;//-2+5x^2y+z-yz+x^2yz

cin >> testpoly1;

cout << "vvedite polynom 2" << endl;// 2 + y - 2z + x ^ 2y

cin >> testpoly2;

cout << "summa polynomov" << endl;

result = testpoly1 + testpoly2;

cout << result << endl;

cout << "raznost polynomov" << endl;

result = testpoly1 - testpoly2;

cout << result << endl;

cout << "proizvedenie polynomov" << endl;

result = testpoly1 \* testpoly2;

cout << result << endl;

}

catch (const char \*k)

{

cout << k;

}

system("pause");

return 0;

## }Приложение 2. Класс TNode

template <typename TKey, typename TData>

struct TNode

{

TKey key;

TData data;

TNode\* pNext;

TNode();

TNode(TKey \_key, TData \_data);

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

~TNode();

const TNode<TKey, TData>& operator=(const TNode<TKey, TData>& \_tnode);

};

template <typename TKey, typename TData>

TNode<TKey, TData>::TNode()

{

key = 0;

pNext = nullptr;

}

template <typename TKey, typename TData>

TNode<TKey, TData>::TNode(TKey \_key, TData \_data)

{

pNext = nullptr;

key = \_key;

data = \_data;

}

template <typename TKey, typename TData>

TNode<TKey, TData>::TNode(const TNode<TKey, TData>& \_tnode)

{

data = \_tnode.data;

pNext = nullptr;

key = \_tnode.key;

}

template<typename TKey, typename TData>

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

{

pNext = nullptr;

}

template<typename TKey, typename TData>

const TNode<TKey, TData>& TNode<TKey, TData>::operator=(const TNode<TKey, TData>& \_tnode)

{

if (this != &\_tnode)

{

key = \_tnode.key;

data = \_tnode.data;

}

return \*this;

}

## Приложение 3. Класс TList

template <typename TKey, typename TData>

class TList

{

protected:

TNode<TKey, TData>\* pFirst;

TNode<TKey, TData>\* pNext;

TNode<TKey, TData>\* pPrevious;

TNode<TKey, TData>\* pCurrent;

public:

TList();

TList(const TList<TKey, TData>& \_tlist);

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

~TList();

void InsertBegin(TKey \_newKey, TData \_data);

void InsertEnd(TKey \_newKey, TData \_data);

void InsertBefore(TKey \_key, TKey \_newKey, TData \_data);

void InsertAfter(TKey \_key, TKey \_newKey, TData \_data);

void InsertBegin(const TNode<TKey, TData>& \_tnode);

void InsertEnd(const TNode<TKey, TData>& \_tnode);

void InsertBefore(TKey \_key, const TNode<TKey, TData>& \_tnode);

void InsertAfter(TKey \_key, const TNode<TKey, TData>& \_tnode);

void Remove(TKey \_key);

void Reset();

void Next();

bool IsEnded() const;

bool IsEmpty() const;

TKey getCurrentNodeKey() const;

TData getCurrentNodeData() const;

};

template <typename TKey, typename TData>

TList<TKey, TData>::TList()

{

pFirst = nullptr;

pPrevious = nullptr;

pNext = nullptr;

pCurrent = nullptr;

}

template <typename TKey, typename TData>

TList<TKey, TData>::TList(const TList& \_tlist)

{

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

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

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

if (tmp != nullptr)

{

while (tmp->pNext != nullptr)

{

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

node = node->pNext;

tmp = tmp->pNext;

}

pPrevious = nullptr;

pCurrent = pFirst;

if (pFirst->pNext != nullptr)

pNext = pFirst->pNext;

}

}

template <typename TKey, typename TData>

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

{

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

pCurrent = pFirst;

pPrevious = nullptr;

pNext = nullptr;

}

template <typename TKey, typename TData>

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

{

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

TNode<TKey, TData> \*next;

while (del != nullptr)

{

next = del->pNext;

delete del;

del = next;

}

}

template <typename TKey, typename TData>

void TList<TKey, TData>::InsertBegin(TKey \_newKey, TData \_data)

{

if (pFirst == nullptr)

{

pFirst = new TNode<TKey, TData>(\_newKey, \_data);

pCurrent = pFirst;

return;

}

TNode<TKey, TData>\* newFirstNode = new TNode<TKey, TData>(\_newKey, \_data);

newFirstNode->pNext = pFirst;

if (pCurrent == pFirst)

pPrevious = newFirstNode;

pFirst = newFirstNode;

}

template <typename TKey, typename TData>

void TList<TKey, TData>::InsertEnd(TKey \_newKey, TData \_data)

{

if (pFirst == nullptr)

{

pFirst = new TNode<TKey, TData>(\_newKey, \_data);

pCurrent = pFirst;

return;

}

TNode<TKey, TData>\* lastNode = new TNode<TKey, TData>(\_newKey, \_data);

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

while (penultimateNode->pNext != nullptr)

{

penultimateNode = penultimateNode->pNext;

}

penultimateNode->pNext = lastNode;

if (pCurrent = penultimateNode)

pNext = lastNode;

}

template <typename TKey, typename TData>

void TList<TKey, TData>::InsertBefore(TKey \_key, TKey \_newKey, TData \_data)

{

if (pFirst == nullptr) return;

if (pFirst->key == \_key)

{

InsertBegin(\_newKey, \_data);

return;

}

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

while ((previousNode->pNext != nullptr) && (previousNode->pNext->key != \_key))

{

previousNode = previousNode->pNext;

}

if (previousNode->pNext == nullptr)

throw "bad";

TNode<TKey, TData>\* nextNode = previousNode->pNext;

TNode<TKey, TData>\* newNode = new TNode<TKey, TData>(\_newKey, \_data);

previousNode->pNext = newNode;

newNode->pNext = nextNode;

if (pCurrent == previousNode)

pNext = newNode;

if (pCurrent == nextNode)

pPrevious = newNode;

}

template <typename TKey, typename TData>

void TList<TKey, TData>::InsertAfter(TKey \_key, TKey \_newKey, TData \_data)

{

if (pFirst == nullptr) return;

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

while ((previousNode != nullptr) && (previousNode->key != \_key))

{

previousNode = previousNode->pNext;

}

if (previousNode == nullptr)

throw "bad";

TNode<TKey, TData>\* nextNode = previousNode->pNext;

TNode<TKey, TData>\* newNode = new TNode<TKey, TData>(\_newKey, \_data);

previousNode->pNext = newNode;

newNode->pNext = nextNode;

if (pCurrent == previousNode)

pNext = newNode;

if (pCurrent == nextNode)

pPrevious = newNode;

}

template<typename TKey, typename TData>

void TList<TKey, TData>::InsertBegin(const TNode<TKey, TData>& \_tnode)

{

InsertBegin(\_tnode.key, \_tnode.data);

}

template<typename TKey, typename TData>

void TList<TKey, TData>::InsertEnd(const TNode<TKey, TData>& \_tnode)

{

InsertEnd(\_tnode.key, \_tnode.data);

}

template<typename TKey, typename TData>

void TList<TKey, TData>::InsertBefore(TKey \_key, const TNode<TKey, TData>& \_tnode)

{

InsertBefore(\_key, \_tnode.key, \_tnode.data);

}

template<typename TKey, typename TData>

void TList<TKey, TData>::InsertAfter(TKey \_key, const TNode<TKey, TData>& \_tnode)

{

InsertAfter(\_key, \_tnode.key, \_tnode.data);

}

template <typename TKey, typename TData>

void TList<TKey, TData>::Remove(TKey \_key)

{

if (pFirst == nullptr) return;

if (pFirst->key == \_key)

{

bool wasFirstNodeCurrent = (pCurrent == pFirst);

bool wasFirstNodePrevious = (pPrevious == pFirst);

TNode<TKey, TData>\* nextNode = pFirst->pNext;

delete pFirst;

pFirst = nextNode;

if (wasFirstNodeCurrent) pCurrent = nullptr;

if (wasFirstNodePrevious) pPrevious = nullptr;

return;

}

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

while ((previousNode->pNext != nullptr) && (previousNode->pNext->key != \_key))

{

previousNode = previousNode->pNext;

}

if (previousNode->pNext == nullptr)

throw "bad";

TNode<TKey, TData>\* nextNode = previousNode->pNext->pNext;

bool wasRemovingNodeCurrent = (pCurrent == previousNode->pNext);

bool wasRemovingNodeNext = (pNext == previousNode->pNext);

delete previousNode->pNext;

previousNode->pNext = nextNode;

if (wasRemovingNodeCurrent)

pCurrent = nullptr;

if (wasRemovingNodeNext)

pNext = nullptr;

}

template <typename TKey, typename TData>

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

{

pCurrent = pFirst;

pPrevious = nullptr;

if (pFirst != nullptr)

pNext = pFirst->pNext;

}

template <typename TKey, typename TData>

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

{

if (pCurrent == nullptr)

return true;

return false;

}

template <typename TKey, typename TData>

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

{

return (pFirst == nullptr);

}

template <typename TKey, typename TData>

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

{

if (IsEnded() == true)

throw "bad";

pPrevious = pCurrent;

pCurrent = pNext;

if (pNext != nullptr)

{

pNext = pNext->pNext;

}

}

template <typename TKey, typename TData>

TKey TList<TKey, TData>::getCurrentNodeKey() const

{

if (pCurrent == nullptr)

{

}

return pCurrent->key;

}

template <typename TKey, typename TData>

TData TList<TKey, TData>::getCurrentNodeData() const

{

if (pCurrent == nullptr)

{

}

return pCurrent->koef;

}

## Приложение 4. Класс Monom

#define Monom TNode<UINT, double>

template<>

struct Monom

{

UINT key;//degree

double koef;//koef

Monom\* pNext;

TNode();

TNode(UINT \_key, double \_koef);

TNode(double \_koef);

TNode(const Monom& \_monom);

~TNode();

bool operator==(const Monom& \_monom) const;

bool operator!=(const Monom& \_monom) const;

const Monom& operator=(const Monom& \_monom);

Monom operator-() const;

Monom operator+(const Monom& \_monom);

Monom operator-(const Monom& \_monom);

Monom operator\*(const Monom& \_monom);

bool operator>(const Monom& \_monom) const;

bool operator<(const Monom& \_monom) const;

char znakmonoma();

friend istream& operator>>(istream& in, Monom& \_monom);

friend ostream& operator<<(ostream& out, const Monom& \_monom);

static Monom convert(const string& \_monom);

};

Monom::TNode()

{

pNext = nullptr;

key = 0;

koef = 0.0;

}

Monom::TNode(UINT \_key, double \_koef)

{

if (\_key > MAX\_KEY)

throw "bad";

key = \_key;

koef = \_koef;

pNext = nullptr;

}

Monom::TNode(double \_koef)

{

key = 0;

koef = \_koef;

pNext = nullptr;

}

Monom::TNode(const Monom& \_monom)

{

if (&\_monom != nullptr)

{

key = \_monom.key;

koef = \_monom.koef;

pNext = nullptr;

}

else

{

key = 0;

koef = 0.0;

pNext = nullptr;

}

}

Monom::~TNode()

{

pNext = nullptr;

}

bool Monom::operator==(const Monom & \_monom) const

{

if ((this->key == \_monom.key) || (this->koef == \_monom.koef))

return true;

return false;

}

bool Monom::operator!=(const Monom & \_monom) const

{

if ((this->key != \_monom.key) || (this->koef != \_monom.koef))

return true;

return false;

}

const Monom& Monom::operator=(const Monom& \_monom)

{

if (\*this != \_monom) // operator!= ==

{

key = \_monom.key;

koef = \_monom.koef;

}

return \*this;

}

Monom Monom::operator-() const

{

return Monom(this->key, this->koef\*(-1));

}

Monom Monom::operator+(const Monom& \_monom)

{

if (key != \_monom.key)

throw "ne te stepeni";

return Monom(key, koef + \_monom.koef);

}

Monom Monom::operator-(const Monom& \_monom)

{

if (key != \_monom.key)

throw "Ne sovpadayut stepeni";

return Monom(key, koef - \_monom.koef);

}

Monom Monom::operator\*(const Monom& \_monom)

{

if ((key / 100 + \_monom.key / 100 > MAX\_DEGREE)

|| ((key % 100) / 10 + (\_monom.key % 100) / 10 > MAX\_DEGREE)

|| (key % 10 + \_monom.key % 10 > MAX\_DEGREE))

throw "Nevernaya stepen";

return Monom(key + \_monom.key, koef \* \_monom.koef);

}

inline bool Monom::operator>(const Monom & \_monom) const

{

if (key > \_monom.key)

return true;

return false;

}

inline bool Monom::operator<(const Monom & \_monom) const

{

if (key < \_monom.key)

return true;

return false;

}

inline char Monom::znakmonoma()

{

if (koef > 0)

return '+';

if (koef < 0)

return '-';

}

Monom Monom::convert(const string & \_monom)

{

UINT key;

double koef;

string new\_monom;

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

if ((\_monom[i] != ' ') && (\_monom[i] != '\*') && (\_monom[i] != '+') && (\_monom[i] != '-'))

new\_monom.push\_back(\_monom[i]);

string coef;

UINT x\_degree = 0;

UINT y\_degree = 0;

UINT z\_degree = 0;

int j = 0;

if ((new\_monom[j] == 'x') || (new\_monom[j] == 'y') || (new\_monom[j] == 'z'))

coef.push\_back('1');

else

while ((new\_monom[j] != 'x') && (new\_monom[j] != 'y')

&& (new\_monom[j] != 'z') && (new\_monom.length() - j))

coef.push\_back(new\_monom[j++]);

if ((!coef.empty()) && (coef.find\_first\_not\_of("0123456789.") == coef.npos))

koef = stod(coef);

else throw "bad1";

for (int i = j; i < new\_monom.length(); i++)

{

switch (new\_monom[i])

{

case 'x':

{

if (new\_monom[i + 1] == '^')

if (('0' <= new\_monom[i + 2]) && (new\_monom[i + 2] <= '9'))

if (x\_degree + new\_monom[i + 2] - '0' > MAX\_DEGREE)

throw "bad2";

else

{

x\_degree += new\_monom[i + 2] - '0';

i += 2;

}

else throw "bad3";

else if ((new\_monom[i + 1] == 'y') || (new\_monom[i + 1] == 'z') || (new\_monom[i + 1] == 0))

if (x\_degree + 1 > MAX\_DEGREE)

throw "bad4";

else x\_degree++;

else if ((i + 1 >= new\_monom.length()) || (new\_monom[i + 1] == ' ')

|| (new\_monom[i + 1] == '+') || (new\_monom[i + 1] == '\*'))

x\_degree++;

else

throw "bad5";

break;

}

case 'y':

{

if (new\_monom[i + 1] == '^')

if (('0' <= new\_monom[i + 2]) && (new\_monom[i + 2] <= '9'))

if (y\_degree + new\_monom[i + 2] - '0' > MAX\_DEGREE)

throw "bad6";

else

{

y\_degree += new\_monom[i + 2] - '0';

i += 2;

}

else throw "bad7";

else if ((new\_monom[i + 1] == 'x') || (new\_monom[i + 1] == 'z') || (new\_monom[i + 1] == 0))

if (y\_degree + 1 > MAX\_DEGREE)

throw "bad8";

else y\_degree++;

else if ((i + 1 >= new\_monom.length()) || (new\_monom[i + 1] == ' ')

|| (new\_monom[i + 1] == '+') || (new\_monom[i + 1] == '\*'))

y\_degree++;

else throw "bad9";

break;

}

case 'z':

{

if (new\_monom[i + 1] == '^')

if (('0' <= new\_monom[i + 2]) && (new\_monom[i + 2] <= '9'))

if (z\_degree + new\_monom[i + 2] - '0' > MAX\_DEGREE)

throw "bad10";

else

{

z\_degree += new\_monom[i + 2] - '0';

i += 2;

}

else throw "bad11";

else if ((new\_monom[i + 1] == 'x') || (new\_monom[i + 1] == 'y') || (new\_monom[i + 1] == 0))

if (z\_degree + 1 > MAX\_DEGREE)

throw "bad12";

else z\_degree++;

else if ((i + 1 >= new\_monom.length()) || (new\_monom[i + 1] == ' ')

|| (new\_monom[i + 1] == '+') || (new\_monom[i + 1] == '\*'))

z\_degree++;

else throw "bad13";

break;

}

default: throw "bad14";

break;

}

}

key = x\_degree \* 100 + y\_degree \* 10 + z\_degree;

return Monom(key, koef);

}

istream & operator>>(istream & in, Monom & \_monom)

{

in.ignore(in.rdbuf()->in\_avail());

string line;

getline(in, line);

int lengthOfExpression = int(line.length());

while (lengthOfExpression)

{

char \_sign = '+';

int start;

int end;

string s\_monom;

int p;

p = 2;

for (int i = 0; i < lengthOfExpression; i++)

{

if (line[i] == ' ');

else if ((line[i] == '+') || (line[i] == '-'))

{

if (p == 0)

throw "bad";

p = 0;

\_sign = line[i];

}

else if ((line[i] == 'x') || (line[i] == 'y') || (line[i] == 'z')

|| (('0' <= line[i]) && (line[i] <= '9')))

{

if (p == 1)

throw "bad";

p = 1;

start = i;

for (end = i; ((line[end] != '+') && (line[end] != '-') && (end < lengthOfExpression)); end++);

break;

}

else throw "bad";

}

s\_monom = line.substr(start, end - start + 1);

\_monom = Monom().convert(s\_monom) \* (44 - \_sign);

break;

}

return in;

}

ostream& operator<<(ostream& out, const Monom& \_monom)

{

if (\_monom.koef < 0)

out << " -";

if ((\_monom.key == 0) || (abs(\_monom.koef) != 1))

out << " " << abs(\_monom.koef);

if (\_monom.key / 100 == 1)

out << (abs(\_monom.koef) != 1 ? " \* x" : " x");

else if (\_monom.key / 100 != 0)

out << (abs(\_monom.koef) != 1 ? " \* x^" : " x^") << \_monom.key / 100;

if ((\_monom.key % 100) / 10 == 1)

out << (abs(\_monom.koef) != 1 || (\_monom.key / 100) ? " \* y" : " y");

else if ((\_monom.key % 100) / 10 != 0)

out << (abs(\_monom.koef) != 1 || (\_monom.key / 100) ? " \* y^" : " y^") << \_monom.key % 100 / 10;

if (\_monom.key % 10 == 1)

out << (abs(\_monom.koef) != 1 || (\_monom.key / 100) || ((\_monom.key % 100) / 10) ? " \* z" : " z");

else if (\_monom.key % 10 != 0)

out << (abs(\_monom.koef) != 1 || (\_monom.key / 100) || ((\_monom.key % 100) / 10) ? " \* z^" : " z^") << \_monom.key % 10;

return out;

};

## Приложение 5. Класс Polynom

class Polynom

{

private:

static Polynom convert(const string& \_expression);

protected:

TList<UINT, double> \*monoms;

public:

Polynom();

Polynom(const string& \_expression);

Polynom(const TList<UINT, double>& \_list);

Polynom(const Monom& \_monom);

Polynom(const Polynom& \_polynom);

~Polynom();

Polynom& operator=(const Polynom& \_polynom);

bool operator==(const Polynom& \_polynom);

Polynom operator-() const;

Polynom operator+(const Polynom& \_polynom) const;

Polynom operator+(const Monom& \_monom) const;

Polynom operator-(const Polynom& \_polynom) const;

Polynom operator-(const Monom& \_monom) const;

Polynom operator\*(const Polynom& \_polynom) const;

Polynom operator\*(const Monom& \_monom) const;

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

friend istream& operator>>(istream& in, Polynom& \_polynom);

};

Polynom Polynom::convert(const string& \_expression)

{

string line = \_expression;

Polynom result;

int lengthOfExpression = int(line.length());

result = result + (0.);

while (lengthOfExpression)

{

char \_sign = '+';

int start\_index;

int end\_index;

string s\_monom;

int prevElement = 0;

for (int i = 0; i < lengthOfExpression; i++)

{

start\_index = i;

end\_index = i;

if (line[i] == ' ');

else if ((line[i] == '+') || (line[i] == '-'))

{

if (prevElement == 1)

throw "bad1";

prevElement = 1;

\_sign = line[i];

}

else if ((line[i] == 'x') || (line[i] == 'y') || (line[i] == 'z')

|| (('0' <= line[i]) && (line[i] <= '9')))

{

if (prevElement == 2)

throw "bad";

prevElement = 2;

while ((line[end\_index] != '+') && (line[end\_index] != '-') && (end\_index < lengthOfExpression))

end\_index++;

break;

}

else throw "bad2";

}

s\_monom = line.substr(start\_index, end\_index - start\_index + 1);

result = result + Monom().convert(s\_monom) \* (44 - \_sign);

if (lengthOfExpression - end\_index <= 0)

break;

line = line.substr(end\_index, lengthOfExpression - end\_index);

lengthOfExpression = int(line.length());

}

return result;

}

Polynom::Polynom()

{

monoms = new TList<UINT, double>;

monoms->InsertBegin(0, .0);

}

Polynom::Polynom(const string& \_expression)

{

\*this = convert(\_expression);

}

Polynom::Polynom(const TList<UINT, double>& \_list)

{

monoms = new TList<UINT, double>();

monoms->InsertEnd(0, .0);

TList<UINT, double> \*tmp\_list = new TList<UINT, double>(\_list);

tmp\_list->Reset();

while (!tmp\_list->IsEnded())

{

if (tmp\_list->getCurrentNodeKey() > MAX\_KEY)

throw "bad3";

\*this = \*this + Monom(tmp\_list->getCurrentNodeKey(), tmp\_list->getCurrentNodeData());

tmp\_list->Next();

}

tmp\_list->Reset();

delete tmp\_list;

}

Polynom::Polynom(const Monom & \_monom)

{

monoms = new TList<UINT, double>();

monoms->InsertBegin(\_monom.key, \_monom.koef);

}

Polynom::Polynom(const Polynom& \_polynom)

{

monoms = new TList<UINT, double>(\*\_polynom.monoms);

}

Polynom::~Polynom()

{

delete monoms;

}

Polynom & Polynom::operator=(const Polynom& \_polynom)

{

if (\*this == \_polynom)

{

return \*this;

}

delete monoms;

monoms = new TList<UINT, double>(\*\_polynom.monoms);

return \*this;

}

bool Polynom::operator==(const Polynom & \_polynom)

{

\_polynom.monoms->Reset();

monoms->Reset();

while (!\_polynom.monoms->IsEnded())

{

if (monoms->IsEnded()

|| (\_polynom.monoms->getCurrentNodeKey() != monoms->getCurrentNodeKey())

|| (\_polynom.monoms->getCurrentNodeData() != monoms->getCurrentNodeData()))

{

monoms->Reset();

\_polynom.monoms->Reset();

return false;

}

\_polynom.monoms->Next();

monoms->Next();

}

if (!monoms->IsEnded())

return false;

monoms->Reset();

\_polynom.monoms->Reset();

return true;

}

Polynom Polynom::operator-() const

{

return (\*this \* (-1.));

}

Polynom Polynom::operator+(const Monom& \_monom) const

{

if (this->monoms->IsEmpty())

{

TList<UINT, double> tmp;

tmp.InsertBegin(\_monom.key, \_monom.koef);

return Polynom(tmp);

}

Polynom result(\*this);

result.monoms->Reset();

UINT currentKey = result.monoms->getCurrentNodeKey();

while ((currentKey > \_monom.key) && (!result.monoms->IsEnded()))

{

result.monoms->Next();

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

currentKey = result.monoms->getCurrentNodeKey();

}

if (currentKey > \_monom.key)

result.monoms->InsertEnd(\_monom.key, \_monom.koef);

if (currentKey == \_monom.key)

{

double newData = result.monoms->getCurrentNodeData() + \_monom.koef;

if (newData != 0.)

result.monoms->InsertAfter(currentKey, \_monom.key, newData);

result.monoms->Remove(currentKey);

}

if (currentKey < \_monom.key)

result.monoms->InsertBefore(currentKey, \_monom.key, \_monom.koef);

result.monoms->Reset();

return result;

}

Polynom Polynom::operator+(const Polynom& \_polynom) const

{

Polynom result;

Polynom p1(\*this);

Polynom p2(\_polynom);

p1.monoms->Reset();

p2.monoms->Reset();

while (!p1.monoms->IsEnded() && !p2.monoms->IsEnded())

{

Monom m1 = Monom(p1.monoms->getCurrentNodeKey(), p1.monoms->getCurrentNodeData());

Monom m2 = Monom(p2.monoms->getCurrentNodeKey(), p2.monoms->getCurrentNodeData());

if (m1.key == m2.key)

{

Monom sum = m1 + m2;

if (sum.koef != 0)

result = result + sum;

p1.monoms->Next();

p2.monoms->Next();

}

else if (m1.key > m2.key)

{

result = result + m2;

p2.monoms->Next();

}

else

{

result = result + m1;

p1.monoms->Next();

}

}

while (!p1.monoms->IsEnded())

{

Monom m1 = Monom(p1.monoms->getCurrentNodeKey(), p1.monoms->getCurrentNodeData());

result = result + m1;

p1.monoms->Next();

}

while (!p2.monoms->IsEnded())

{

Monom m2 = Monom(p2.monoms->getCurrentNodeKey(), p2.monoms->getCurrentNodeData());

result = result + m2;

p2.monoms->Next();

}

return result;

}

Polynom Polynom::operator-(const Polynom & \_polynom) const

{

return (Polynom(\*this) + (-\_polynom));

}

Polynom Polynom::operator-(const Monom& \_monom) const

{

return (Polynom(\*this) + Monom(\_monom.key, -\_monom.koef));

}

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

{

Polynom result;

Polynom polynomial(\_polynom);

polynomial.monoms->Reset();

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

{

double currentData = polynomial.monoms->getCurrentNodeData();

UINT currentKey = polynomial.monoms->getCurrentNodeKey();

Polynom tmp(Polynom(\*this) \* Monom(currentKey, currentData));

result = result + tmp;

polynomial.monoms->Next();

}

return result;

}

Polynom Polynom::operator\*(const Monom& \_monom) const

{

Polynom result;

Polynom polynomial(\*this);

if (\_monom.koef == 0)

return result;

polynomial.monoms->Reset();

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

{

Monom tmp(polynomial.monoms->getCurrentNodeKey(), polynomial.monoms->getCurrentNodeData());

tmp = tmp \* \_monom;

if (tmp.koef != 0)

result = result + tmp;

polynomial.monoms->Next();

}

return result;

}

ostream& operator<<(ostream& out, const Polynom& \_polynom)

{

\_polynom.monoms->Reset();

if (\_polynom.monoms->IsEmpty())

out << " 0 ";

else

{

if (\_polynom.monoms->getCurrentNodeData() == 0)

\_polynom.monoms->Next();

if (\_polynom.monoms->IsEnded())

{

out << " 0 ";

return out;

}

Monom tmp(\_polynom.monoms->getCurrentNodeKey(), \_polynom.monoms->getCurrentNodeData());

out << " ";

if (tmp.znakmonoma() == '-')

out << "-";

if ((tmp.key == 0) || (abs(tmp.koef) != 1))

out << " " << abs(tmp.koef);

if (tmp.key / 100 == 1)

out << (abs(tmp.koef) != 1 ? " \* x" : " x");

else if (tmp.key / 100 != 0)

out << (abs(tmp.koef) != 1 ? " \* x^" : " x^") << tmp.key / 100;

if ((tmp.key % 100) / 10 == 1)

out << (abs(tmp.koef) != 1 || (tmp.key / 100) ? " \* y" : " y");

else if ((tmp.key % 100) / 10 != 0)

out << (abs(tmp.koef) != 1 || (tmp.key / 100) ? " \* y^" : " y^") << tmp.key % 100 / 10;

if (tmp.key % 10 == 1)

out << (abs(tmp.koef) != 1 || (tmp.key / 100) || ((tmp.key % 100) / 10) ? " \* z" : " z");

else if (tmp.key % 10 != 0)

out << (abs(tmp.koef) != 1 || (tmp.key / 100) || ((tmp.key % 100) / 10) ? " \* z^" : " z^") << tmp.key % 10;

\_polynom.monoms->Next();

while (!\_polynom.monoms->IsEnded())

{

Monom tmp = Monom(\_polynom.monoms->getCurrentNodeKey(), \_polynom.monoms->getCurrentNodeData());

if (tmp.znakmonoma() == '+')

out << " +";

out << tmp;

\_polynom.monoms->Next();

}

}

return out;

}

istream& operator>>(istream& in, Polynom& \_polynom)

{

delete \_polynom.monoms;

\_polynom.monoms = new TList<UINT, double>;

string line;

getline(in, line);

\_polynom = Polynom::convert(line);

return in;

}