МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
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

Направление подготовки: «Программная инженерия»

ОТЧЕТ

по лабораторной работе №5

Аналитические преобразования полиномов от нескольких переменных (списки)

Выполнил: студент группы
3822Б1ПР2

_____В. Е. Филатьев
Подпись

Нижний Новгород
2023

Содержание

1.Введение:

Многочлен фундаментальное понятие в алгебре и математическом анализе. В простейшем случае многочленом называется функция вещественной или комплексной переменной следующего вида:

$P(x) = c_0 + c_1x^1 + c_2x^2 + \cdots + c_nx^n$, где ci фиксированные коэффициенты.

Максимальная степень ci среди слагаемых-одночленов называется степенью многочлена.

Примеры:

- x2 + 2x + 1 (многочлен второй степени)
- x5 – 12x3 + 4 (многочлен пятой степени)

В более общем случае многочлен может содержать степени нескольких независимых переменных например:

x^2y + 2x – 2y. Это многочлен от двух переменных (x,y) второй степени.

Многочлены как функции можно складывать, перемножать, а в некоторых случая и делить один на другой. Коэффициенты многочлена могут быть не обязательно числовыми.

Допускается многочлен, вообще не содержащий переменных, то есть числовая константа: P(x) = c ; его степень считается равной нулю. Исключением является нулевой многочлен, тождественно равный нулю: P(x) = 0, его степень не определяется (иногда считается равной -1 или $-\infty$).

Цель лабораторной работы заключается в изучении методов обработки полиномов с использованием компьютера. Для этого исследуются различные способы хранения полиномов и разрабатываются программы для их обработки. Основной учебной целью работы является практическое освоение методов организации структур данных с использованием списков. В ходе выполнения лабораторной работы разрабатывается общая форма представления линейных списков и программы для работы со списками, которые могут быть применены и в других областях приложений.

2. Постановка задачи:

В рамках лабораторной работы ставится задача создания программных средств, поддерживающих эффективное хранение полиномов и выполнение основных операций над ними:

- Формирование полинома из строки
- Вывод полинома в строку
- Вычисление полинома в заданной точке
- Сложение/вычитание полиномов
- Умножение полинома на константу

Класс может поддерживать дополнительные операции:

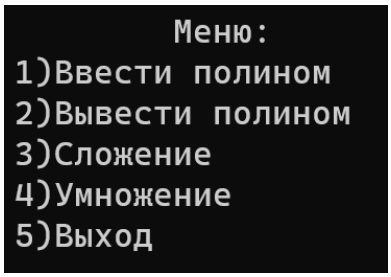
- Умножение полиномов
- Деление полиномов
- Дифференцирование полинома по данной переменной

Предполагается, что в качестве структуры хранения будут использоваться списки. В качестве дополнительной цели в лабораторной работе ставится также задача разработки некоторого общего представления списков и операций по их обработке. В числе операций над списками должны быть реализованы следующие действия:

- Поддержка понятия текущего звена;
- Вставка звеньев в начало, после текущей позиции и в конец списков;
- Удаление звеньев в начале и в текущей позиции списков;
- Организация последовательного доступа к звеньям списка (итератор).

3. Руководство пользователя

При начале работы программы пользователю выводится главное меню (*Рис 1*). Пользователь может создать полином (Рис 2). Вывести полином (Рис 3). Сложить с мономом или полиномом (Рис 4). Умножить на полином или моном (Рис 5).



Рису 1

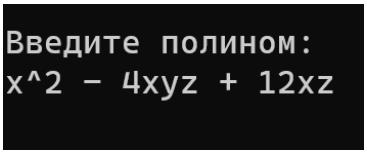


Рис 2

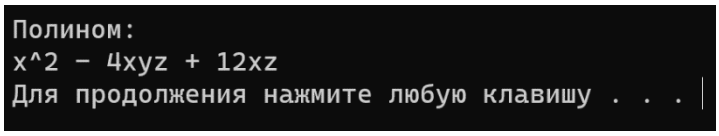


Рис 3

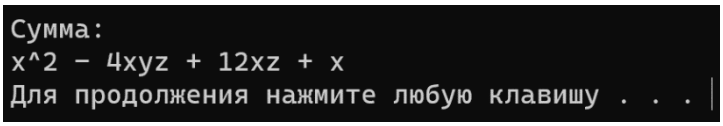


Рис 4

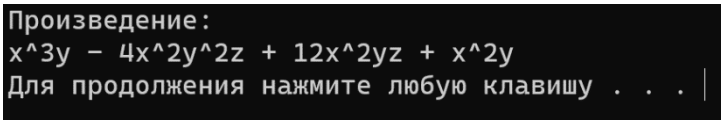
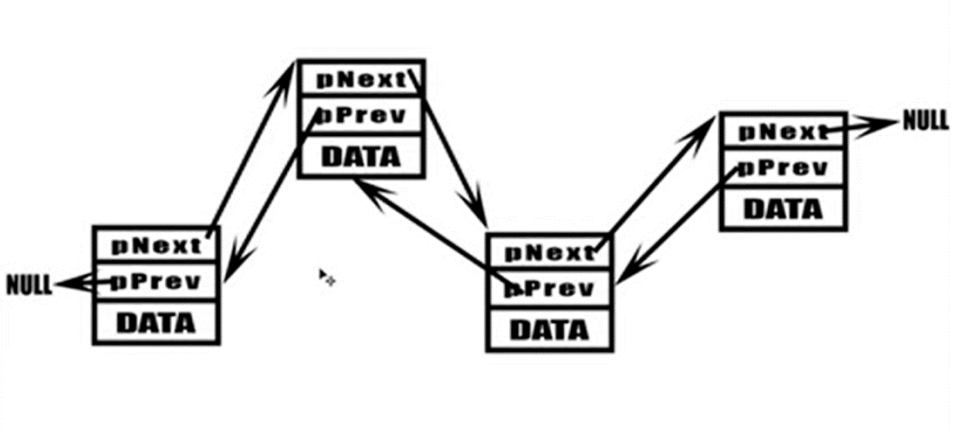


Рис 5

4. Руководство программиста:

4.1. Описание структур:

Очередь, построенная на двухсвязном списке в языке программирования C++, представляет собой структуру данных, которая поддерживает операции добавления элемента в конец очереди (push) и удаления элемента из начала очереди (pop). Двухсвязной список состоит из узлов, каждый из которых содержит указатели на предыдущий и следующий элементы. Это позволяет эффективно добавлять и удалять элементы как в начале, так и в конце списка.



Для реализации очереди на двухсвязном списке в C++ необходимо создать класс, который будет содержать методы для добавления и удаления элементов, а также указатели на начало и конец списка. Метод push будет добавлять новый элемент в конец списка, а метод pop будет удалять элемент из начала списка. Преимуществом использования двухсвязного списка для реализации очереди является то, что операции добавления и удаления элементов выполняются за константное время O(1). Однако, необходимо учитывать, что использование двухсвязного списка требует больше памяти для хранения указателей на предыдущий и следующий элементы. Таким образом, очередь, построенная на двухсвязном списке в C++, является эффективной структурой данных для реализации очереди с быстрым доступом к началу и концу списка.

4.2.Описание программы:

Программа состоит классов, отвечавших за работу со списком и классы моном и полином:

Класс список:

Класс список – шаблонный класс с изменяемым типом хранения содержащий:

Внутренний класс:

- class Node; - Данный класс хранит значение ячейки (T value;) и указатель на следующую и предыдущую ячейку (Node* pNext, pPrev). В классе перегружен конструктор инициализации.
- Class iterator; - Данный класс предназначит для обхода списка по элементам

Поля:

- Node<T>* head; - указатель на первый элемент списка
- Node<T>* tail; - указывает на последний элемент списка
- size_t size; - размер очереди

Методы:

- List(); - конструктор по умолчанию
- bool empty(); - метод проверки на пустоту списка
- void push_back(T value); - добавление элемента в конец списка
- void push_front(T value); - добавление элемента в начала списка
- void insert(T value, size_t index); - добавление элемента в список по индексу
- void pop_back(); - удаление элемента из конца списка
- void pop_front(); - удаление элемента из начала списка
- void erase(size_t index) – удаление элемента из ячейки по индексу
- void clear(); - очистка списка
- size_t Size() – метод возвращает размер стека
- T& top(); - метод возвращает верхний элемент стека
- ~ List; - деструктор
- Note<T>* begin() – возвращает указатель на первый элемент

- Note<T>* end() – возвращает указатель на следующий элемент конец списка
- T& operator[](size_t index); - метод получение элемента из списка
- T& at(size_t index); - метод получение элемента из списка с проверкой
- T& back(); - метод получить последнего элемента
- T& front(); -метод получения первого элемента
- bool operator==(const List<T> other); - Перегруженный оператор сравнения двух списков
- List(List<T>& other); - Конструктор копирования
- List<T>& operator=(List<T>& other); - Оператор присваивания

Класс моном:

Поля:

- double coef; - Коэффициент монома
- int index; - Индекс (свертка степеней)

Методы:

- TMonom() – Конструктор по умолчанию
- TMonom(string s) – Конструктор инициализации
- TMonom(double coef, int degX = 0, int degY = 0, int degZ = 0) – Конструктор инициализации
- void SetCoef(int cval) – установление коэффициента
- int GetCoef(void) – получение коэффициента
- void SetIndex(int ival) – установление индекса
- int GetIndex(void) – получение индекса
- bool operator==(const TMonom& other) – оператор сравнения
- bool operator<(const TMonom& other) – оператор сравнения

Класс полином:

Поля:

- List<TMonom> list; - список мономов

Методы:

- TPolinom(TPolinom& other) – конструктор копирования
- TPolinom() – конструктор по умолчанию
- TPolinom(string str); - конструктор инициализации
- TPolinom& operator=(TPolinom& other); оператор присваивание
- TPolinom& operator+(TPolinom& q); оператор сложение полиномов
- void setPolinom(string s); - установление полинома из строки
- void operator+(TMonom newMonom); метод добавленния монома
- TPolinom operator*(TMonom monom); метод умножение мономов
- TPolinom operator*(double coef); метод умножение полинома на число
- TPolinom operator* (TPolinom& other); метод умножение полиномов
- bool operator==(TPolinom& other); метод сравнение полиномов на равенство
- string ToString(); метода перевода в строку

5.Результаты:

Тесты:

1) $P = 3x^5y^2z^5 – 5x^4y^3z^3 + 7x^3y^5z$

   $Q = 4x^3y^2z^6 - 6x^2yz^8$

   $P + Q = 3x^5y^2z^5 – 5x^4y^3z^3 + 7x^3y^5z + 4x^3y^2z^6 - 6x^2yz^8$

2) $P = 3x^5y^2z^5 – 5x^4y^3z^3 + 7x^3y^5z$

   $Q = 4x^7y^2z^6 - 6x^6yz^8$

   $P + Q = 3x^5y^2z^5 – 5x^4y^3z^3 + 7x^3y^5z + 4x^7y^2z^6 - 6x^6yz^8$

3) $P = 3x^5y^2z^5 – 5x^4y^3z^3 + 7x^3y^5z$

   $Q = 4x^5y^2z^5 + 5x^4y^3z^3$

   $P + Q = 7x^5y^2z^5 + 7x^3y^5z$

4) $P = 3x^5y^2z^5 – 5x^4y^3z^3 + 7x^7y^5z$

   $Q = 4x^6y^2z^6 - 6x^2yz^8$

   $P + Q = 3x^5y^2z^5 – 5x^4y^3z^3 + 7x^7y^5z + 4x^6y^2z^6 - 6x^2yz^8$

5) $P = 3x^5y^2z^5 – 5x^4y^3z^3 + 7x^7y^5z$

   $Q = -3x^5y^2z^5 + 5x^4y^3z^3 - 7x^7y^5z$

   $P + Q = 0$

6.Заключение

В ходе выполнения этой лабораторной работы, я успешно справился со всеми поставленными задачами. После изучения темы был реализован класс список на основе двухсвязного списка, после в него были добавлены различные улучшения такие как, интегрируемость по списку. Реализовали тесты для проверки основных функций класса. После был реализованы классы моном и полином, и тесты для проверки основных методов класса. Классы были оптимизированы для эффективной работы. Для проверки работоспособности системы были реализованы консольный интерфейс. В интерфейсе проверили работоспособность приложений с работой полиномов.

7.Литература:

1. Википедия/Полином - https://ru.wikipedia.org/wiki/Многочлен
2. Википедия/Список - https://ru.wikipedia.org/wiki/Связный_список
3. Учебно-методическое пособие - https://drive.google.com/drive

8.Приложение:

Класс список:

```
#pragma once
#include "TNode.h"
#include <iostream>
using namespace std;

template<class T>
class List {
	Node<T>* head;
	Node<T>* tail;
	size_t size;

	Node<T>* getNode(int index);
public:
	List();
	~List();
	void push_back(T value);
	void push_front(T value);
	void insert(T value, size_t index);
	void pop_back();
	void pop_front();
	void erase(size_t index);
	void clear();
	bool isEmpty() { return size == 0; }
	size_t Size() { return size; }
	T& operator[](size_t index);
	T& at(size_t index);
	T& back() { return tail->value; }
	T& front() { return head->value; }
	bool operator==(const List<T> other);



public:
	class iterator {
		Node<T>* t;
	public:
		iterator() { t = nullptr; }
		void operator=(Node<T>* note) {
			t = note;
		}
		T& operator*() {
			return t->value;
		}
		void operator++(int k) {
			t = t->pNext;
		}
		void operator--(int k) {
			t = t->pPrev;
		}
		bool operator!=(const Node<T>* note) {
			return t != note;
		}
		Node<T>* getNode() {
			return t;
		}
	};
```

```
	Node<T>* begin() { return head; }
	Node<T>* end() { return (tail == nullptr) ? tail : tail->pNext; }

	List(List<T>& other);
	List<T>& operator=(List<T>& other);

	void insert(T value, List<T>::iterator it);
};

template<class T>
inline Node<T>* List<T>::getNode(int index)
{
	Node<T>* temp;
	if (index < size / 2) {
		int kol = 0;
		temp = head;
		while (kol != index) {
			temp = temp->pNext;
			kol++;
		}
	}
	else {
		int kol = size - 1;
		temp = tail;
		while (kol != index) {
			temp = temp->pPrev;
			kol--;
		}
	}
	return temp;
}

template<class T>
List<T>::List() {
	head = nullptr;
	tail = nullptr;
	size = 0;
}

template<class T>
inline List<T>::List(List<T>& other):List()
{
	List<T>::iterator it;
	it = other.begin();
	while (it != other.end()) {
		this->push_back(*it);
		it++;
	}
}

template<class T>
inline List<T>& List<T>::operator=(List<T>& other)
{
	if (this == &other) {
		return *this;
	}
	this->clear();
	List<T>::iterator it;
	it = other.begin();
	while (it != other.end()) {
		this->push_back(*it);
		it++;
```

```
		}
		return *this;
}

template<class T>
inline List<T>::~List()
{
		clear();
}

template<class T>
inline void List<T>::push_back(T value)
{
		if (size == 0) {
				head = new Node<T>(value);
				tail = head;
		}
		else {
				tail->pNext = new Node<T>(value,tail);
				tail = tail->pNext;
		}
		size++;
}

template<class T>
inline void List<T>::push_front(T value)
{
		if (size == 0) {
				head = new Node<T>(value);
				tail = head;
		}
		else {
				Node<T>* temp = new Node<T>(value, nullptr, head);
				head->pPrev = temp;
				head = temp;
		}
		size++;
}

template<class T>
inline void List<T>::insert(T value, size_t index)
{
		if (index > size)
				throw exception("of range");
		if (index == 0)
				push_front(value);
		else if (index == size)
				push_back(value);
		else {
				Node<T>* temp = getNode(index - 1);
				Node<T>* newNode = new Node<T>(value, temp, temp->pNext);
				temp->pNext->pPrev = newNode;
				temp->pNext = newNode;

				size++;
		}
}

template<class T>
inline void List<T>::insert(T value, List<T>::iterator it)
{
		if (it.getNode() == nullptr)
```

```
                push_back(value);
        else if (it.getNode()->pPrev == nullptr)
                push_front(value);
        else {
                Node<T>* temp = it.getNode()->pPrev;
                Node<T>* newNode = new Node<T>(value, temp, temp->pNext);
                temp->pNext->pPrev = newNode;
                temp->pNext = newNode;

                size++;
        }
}

template<class T>
inline void List<T>::pop_back()
{
        Node<T>* temp = tail->pPrev;
        delete tail;
        tail = temp;
        if (temp != nullptr)
                tail->pNext = nullptr;
        size--;
}

template<class T>
inline void List<T>::pop_front()
{
        Node<T>* temp = head;
        if (head != nullptr) {
                head = head->pNext;
        }
        if (head != nullptr) {
                head->pPrev = nullptr;
        }
        else {
                tail = head;
        }
        delete temp;
        size--;
}

template<class T>
inline void List<T>::erase(size_t index)
{
        if (index > size - 1)
                throw exception("of range");
        if (index == 0)
                pop_front();
        else if (index == size - 1)
                pop_back();
        else {
                Node<T>* temp = getNode(index - 1);
                Node<T>* del = temp->pNext;
                del->pNext->pPrev = temp;
                temp->pNext = del->pNext;

                delete del;
                size--;
        }

}
```

```
template<class T>
inline void List<T>::clear()
{
	while (!isEmpty()) {
		this->pop_back();
	}
	head = tail = nullptr;
}

template<class T>
inline T& List<T>::operator[](size_t index)
{
	return getNode(index)->value;
}

template<class T>
inline T& List<T>::at(size_t index)
{
	if (index > size)
		throw exception("of range");
	return this->operator[](index);
}

template<class T>
inline bool List<T>::operator==(List<T> other)
{
	if (this->size != other.size)
		return false;
	List<T>::iterator it1,it2;
	it1 = this->begin();
	it2 = other.begin();

	while (it1 != this->end()) {
		if (!((*it1) == (*it2)))
			return false;
		it1++;
		it2++;
	}
	return true;
}
```

Класс моном:

```
#pragma once
#include <string>
#include <stack>
using namespace std;
bool CheckNumber(char c)
{
	return (c >= '0' && c <= '9' || c == '.');
}
bool CheckChar(char c) {
	return (c >= 'A' && c <= 'Z' || c >= 'a' && c <= 'z' || c == '_');
}
bool CheckRes(char c) {
	return c == '-';
}
double ToDouble(string s, int& i) {
	double k = 0;
	int kol = 0;
	while (s[i] != '\0' && CheckNumber(s[i])) {
		if (s[i] != '.')
			k = k * 10 + (s[i] - 48);
		if (s[i] == '.' || kol != 0)
			kol++;
		i++;
	}
	i--;
	return k / pow(10, ((kol == 0) ? 0 : kol - 1));
}

string ToStringVar(string s, int& i) {
	string temp;
	char c = s[i];
	while (c != '\0' && CheckChar(c)) {
		temp += s[i];
		i++;
		c = s[i];
	}
	i--;
	return temp;
}

struct TMonom
{
	double coef; // коэффициент монома
	int index; // индекс (свертка степеней)



	TMonom() {
		this->coef = 1;
		this->index = 0;
	}

	TMonom(string s) :TMonom(){
		stack<string> st;
		int i = 0;
		while (i < s.size()) {
			if (CheckRes(s[i])) {
				st.push("-");
			}
```

```
                else if (CheckNumber(s[i])) {
                        double temp = ToDouble(s, i);
                        if (st.empty())
                                coef = temp;
                        else if (st.top() == "-") {
                                coef = -temp;
                                st.pop();
                        }
                        else if (st.top() == "^") {
                                st.pop();
                                if (st.top() == "x" || st.top() == "X")
                                        index += 100 * temp;
                                if (st.top() == "y" || st.top() == "Y")
                                        index += 10 * temp;
                                if (st.top() == "z" || st.top() == "Z")
                                        index += temp;
                                st.pop();
                        }

                }
                else if (CheckChar(s[i])) {
                        string var; var += s[i];
                        if (st.empty())
                                st.push(var);
                        else if (st.top() == "-") {
                                coef = -1;
                                st.pop();
                                st.push(var);
                        }
                        else {
                                if (st.top() == "x" || st.top() == "X")
                                        index += 100;
                                if (st.top() == "y" || st.top() == "Y")
                                        index += 10;
                                if (st.top() == "z" || st.top() == "Z")
                                        index += 1;
                                st.pop();
                                st.push(var);
                        }
                }
                else if (s[i] == '^') {
                        st.push("^");
                }
                i++;
        }
        if (!st.empty()) {
                if (st.top() == "x" || st.top() == "X")
                        index += 100;
                if (st.top() == "y" || st.top() == "Y")
                        index += 10;
                if (st.top() == "z" || st.top() == "Z")
                        index += 1;
                st.pop();
        }

}

TMonom(double coef, int degX = 0, int degY = 0, int degZ = 0) {
        if (degX > 9 || degY > 9 || degZ > 9)
                throw exception("deg > 9");
        this->coef = coef;
        this->index = 100 * degX + 10 * degY + degZ;
```

```
}

void SetCoef(int cval) { coef = cval; }
int GetCoef(void) { return coef; }

void SetIndex(int ival) { index = ival; }
int GetIndex(void) { return index; }


bool operator==(const TMonom& other) {return  this->index == other.index; }

bool operator>(const TMonom& other) {
	if (this->index / 100 == other.index / 100) {
		if ((this->index % 100) / 10 == (other.index % 100) / 10){
			if (this->index % 10 == other.index % 10) {
				return false;
			}
			else
				return this->index % 10 > other.index % 10;
		}
		else
			return (this->index % 100) / 10 > (other.index % 100) / 10;
	}
	else
		return this->index / 100 > other.index / 100;
}

bool operator<(const TMonom& other) {
	return  !(this->operator>(other) && this->operator==(other));
}

TMonom& operator*(const TMonom& other) {
	if (this->index / 100 + other.index / 100 > 9)
		throw exception("of range");
	if ((this->index % 100) / 10 + (other.index % 100) / 10 > 9)
		throw exception("of range");
	if (this->index % 10 + other.index % 10 > 9)
		throw exception("of range");
	TMonom temp;
	temp.index = this-> index + other.index;
	temp.coef = this->coef * other.coef;
	return temp;
}
};
```

Класс полином:

```
#pragma once
#include "TList.h"
#include "TMonom.h"
#include <string>

const int nonDisplayedZeros = 4; // Количество неотображаемых нулей при выводе коэффициента полинома

string RemoveSpace(string s);


class TPolinom
{
public:
	List<TMonom> list;


	TPolinom(TPolinom& other);
	TPolinom() {};
	TPolinom(string str);
	TPolinom& operator=(TPolinom& other); // присваивание
	TPolinom& operator+(TPolinom& q); // сложение полиномов
	void setPolinom(string s);

	// дополнительно можно реализовать:
	void operator+(TMonom newMonom); // добавление монома
	TPolinom operator*(TMonom monom); // умножение мономов
	TPolinom operator*(double coef); // умножение полинома на число
	TPolinom operator* (TPolinom& other); // умножение полиномов
	bool operator==(TPolinom& other); // сравнение полиномов на равенство
	string ToString(); // перевод в строку
};

void TPolinom::setPolinom(string s) {
	list.clear();
	s = RemoveSpace(s);
	string temp;
	int i = 0;
	while (i < s.size()) {
		if (s[i] == '+') {
			this->operator+(TMonom(temp));
			temp.clear();
		}
		else if (s[i] == '-') {
			if (!temp.empty())
				this->operator+(TMonom(temp));
			temp.clear();
			temp.push_back('-');
		}
		else {
			temp += s[i];
		}
		i++;
	}
	if (!temp.empty()) {
		this->operator+(TMonom(temp));
	}
}

TPolinom::TPolinom(TPolinom& other)
{
```

```
	this->list = other.list;
}

TPolinom::TPolinom(string str)
{
	setPolinom(str);
}

TPolinom& TPolinom::operator=(TPolinom& other)
{
	this->list = other.list;
	return *this;
}

void TPolinom::operator+(TMonom m)
{
	List<TMonom>::iterator it;
	it = list.begin();
	bool flag = true;
	while (it != list.end()) {
		if (m > *it)
			break;
		if (*it == m) {
			(*it).coef += m.coef;
			flag = false;
			break;
		}
		it++;
	}
	if (flag && m.coef != 0)
		list.insert(m, it);
}


TPolinom& TPolinom::operator+(TPolinom& other)
{
	List<TMonom>::iterator it;
	it = other.list.begin();
	while (it != other.list.end()) {
		this->operator+(*it);
		it++;
	}

	return *this;
}


TPolinom TPolinom::operator*(TMonom monom)
{
	TPolinom temp = *this;
	List<TMonom>::iterator it;
	it = temp.list.begin();
	while (it != temp.list.end()) {
		(*it) = (*it).operator*(monom);
		it++;
	}

	return temp;
}

TPolinom TPolinom::operator*(double coef)
{
```

```
	List<TMonom>::iterator it;
	it = list.begin();
	while (it != list.end()) {
		(*it).coef *= coef;
		it++;
	}
	return *this;
}

TPolinom TPolinom::operator*(TPolinom& other)
{
	TPolinom temp;
	List<TMonom>::iterator it;
	it = other.list.begin();
	while (it != other.list.end()) {
		TPolinom t = this->operator*(*it);
		temp = temp + t;
		it++;
	}
	*this = temp;
	return *this;
}

bool TPolinom::operator==(TPolinom& other)
{
	return this->list == other.list;
}


string TPolinom::ToString()
{
	string result;
	List<TMonom>::iterator it;
	it = list.begin();
	while (it != list.end()) {
		if (abs((*it).coef) != 1){
			result +=  std::to_string(abs((int)(*it).coef));
		}
		if ((*it).index / 100 != 0) {
			result += "x";
			if((*it).index / 100 != 1)
				result += "^" + std::to_string((*it).index / 100);
		}
		if ((*it).index % 100 / 10 != 0) {
			result += "y";
			if((*it).index % 100 / 10 != 1)
				result += "^" + std::to_string((*it).index % 100 / 10);
		}
		if ((*it).index % 10 != 0) {
			result += "z";
			if ((*it).index % 10 != 1)
				result += "^" + std::to_string((*it).index % 10);
		}
		it++;
		if (it != list.end()) {
			if ((*it).coef < 0)
				result += " - ";
			else
				result += " + ";
		}
	}
	return result;
```

```
}



string RemoveSpace(string s)
{
	int i = 0;
	while (i < s.size()) {
		if (s[i] == ' ')
			s.erase(i, 1);
		else
			i++;
	}
	return s;
}
```

main:

```
#include <iostream>
#include <conio.h>
#include "TList.h"
#include "TPolinom.h"
using namespace std;

int vibor(int kol) {
	char a;
	do {
		a = _getch() - 48;
	} while (!(a >= 1 && a <= kol));
	return (int)a;
}

int main()
{
	setlocale(LC_ALL, "ru");
	int wibor;
	TPolinom polinom;
	do {
		system("cls");
		cout << "\tМеню:\n1)Ввести полином\n2)Вывести
полином\n3)Сложение\n4)Умножение\n5)Выход\n";
		wibor = vibor(5);
		system("cls");
		if (wibor == 1) {
			cout << "Введите полином:" << endl;
			string s; getline(cin, s);
			polinom.setPolinom(s);
		}
		if (wibor == 2) {
			cout << "Полином:" << endl;
			cout << polinom.ToString() << endl;
			system("pause");
		}
		if (wibor == 3) {
			cout << "Введите полином:" << endl;
			string s; getline(cin, s);
			TPolinom temp(s);
			system("cls");
			cout << "Сумма:\n";
			polinom + temp;
			cout << polinom.ToString() << endl;
			system("pause");
		}
		if (wibor == 4) {
			cout << "Введите полином:" << endl;
			string s; getline(cin, s);
			TPolinom temp(s);
			system("cls");
			cout << "Произведение:\n";
			polinom * temp;
			cout << polinom.ToString() << endl;
			system("pause");
		}


	} while (wibor != 5);
```

```
}
```