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

_______Е. А. Соловьева
Подпись

Нижний Новгород
2023

Содержание

1.Введение:

Под полиномом от одной переменной понимается выражение вида (рис.1):

$$P_n(x)=a_nx^n+a_{n-1}x^{n-1}+\ldots+a_1x^1+a_0$$ (Рис.1)

или в более общем виде (рис.2)

$$P_n(x) = \sum_{i=0}^{n} a_i x^i$$ (Рис.2)

где:

n – степень полинома;

$a_i, 0 \le i \le n$ – коэффициенты полинома (действительные или комплексные числа).

Полином можно определить также как выражение из нескольких термов, соединенных знаками сложения или вычитания. Терм включает коэффициент и моном, содержащий одну или несколько переменных, каждая из которых может иметь степень (рис.3)

$$P(x,y,z) = \sum_{i,\,j,\,k} a_{ijk}\, x^i\, y^j\, z^k$$ (Рис.3)

Как пример, полином от трех переменных может иметь вид (рис.4)

$$P(X, Y, Z)=3x^3z - 2y^2z^2 + 3.$$ (Рис. 4)

Подобными называют два (или более) мономов, имеющих одинаковые степени при неизвестных. В число основных операций над полиномами входят действия по вычислению значений полинома при заданных значениях переменных, а также большинство известных математических операций (сложение, вычитание, вычисление частных производных, интегрирование и т.п.). Полиномы как формальный объект хорошо изучены в математике. Математическая модель данной предметной области – алгебра полиномов.

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:

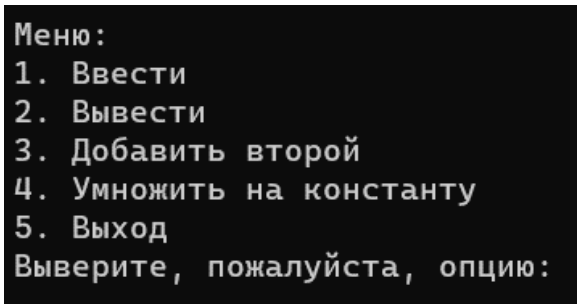
- Умножение полиномов
- Деление полиномов
- Дифференцирование полинома по данной переменной

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).

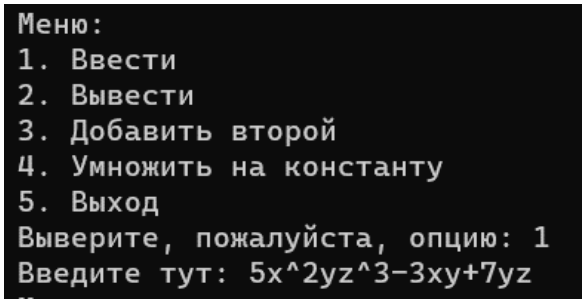
3. Руководство пользователя

При начале работы программы пользователю выводится главное меню (рис.5):



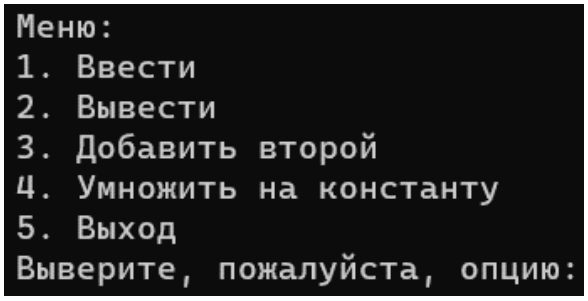
(Рис. 5)

Пользователь выбирает первый пункт и вводит полином (рис.6):



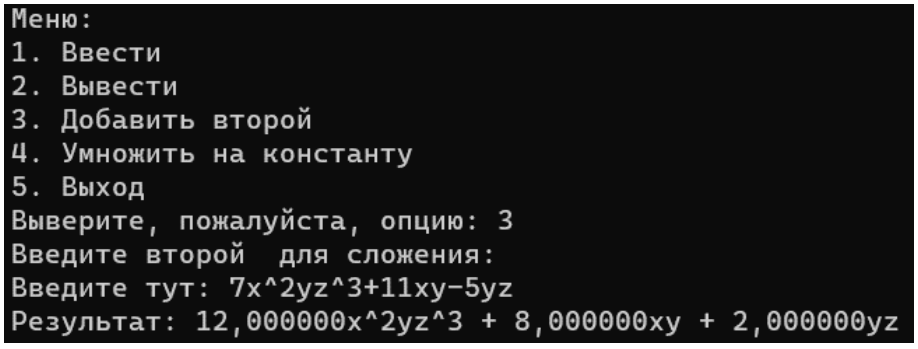
(Рис. 6)

Далее пользователь выбирает действие (рис.7):



(Рис. 7)

Либо перезаписать исходный полином, либо вывести его на экран, либо сложить со вторым полиномом (рис.8), который позже сам пользователь введет, либо умножить исходный полином на константу (рис.9), либо выйти из программы.



(Рис. 8)

```
Результат: 12,000000x^2yz^3 + 8,000000xy + 2,000000yz
Меню:
1. Ввести
2. Вывести
3. Добавить второй
4. Умножить на константу
5. Выход
Выверите, пожалуйста, опцию: 4
Введите константу для умножения: 11
Результат: 132,000000x^2yz^3 + 88,000000xy + 22,000000yz
```
(Рис. 9)

Далее пользователь заканчивает свою работу в программе и выбирает пункт 5 (рис.10):

```
Меню:
1. Ввести
2. Вывести
3. Добавить второй
4. Умножить на константу
5. Выход
Выверите, пожалуйста, опцию: 5
Выход
```
(Рис. 10)

4. Руководство программиста:

4.1. Описание структур:

Для представления полиномов могут быть выбраны различные структуры хранения. Критериями выбора структуры хранения являются размер требуемой памяти и сложность (трудоемкость) реализации операций над полиномами.

Возможный вариант структуры хранения – использование массивов (в случае полиномов от трех переменных – трехмерная матрица коэффициентов полинома). Такой способ обеспечивает простую реализацию операций над полиномами, но он не эффективен в части объема требуемой памяти. Так, при сделанных допущениях для хранения одного полинома в массиве потребуется порядка 8000 байт памяти – при этом в памяти будут храниться в основном параметры мономов с нулевыми коэффициентами.

Разработка более эффективной структуры хранения должна быть выполнена с учетом следующих рекомендаций:

- в структуре хранения должны храниться данные только для мономов с ненулевыми коэффициентами;
- порядок размещения данных в структуре хранения должен обеспечивать возможность быстрого поиска мономов с заданными свойствами (например, для приведения подобных).

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 $X^{A1}Y^{B1}Z^{C1}$ предшествует моному $X^{A2}Y^{B2}Z^{C2}$ тогда и только тогда, если $(A_1 > A_2) \vee (A_1 = A_2) \& (B_1 > B_2) \vee (A_1 = A_2) \& (B_1 = B_2) \& (C_1 > C_2)$

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.

T value;
TNode* pNext;

TList.h:

Поля:

TNode<T>* pFirst; // первое звено

TNode<T>* pCurrent; // текущее звено

TNode<T>* pPrevious; // звено перед текущим

TNode<T>* pLast; // последнее звено

TNode<T>* pStop; // значение указателя, означающего конец списка

int length; // количество звеньев в списке

Методы:

TList();

~TList();

int GetLength() { return length; }

bool IsEmpty(); // список пуст ?

// вставка звеньев

void InsertFirst(T item); // перед первым

void InsertCurrent(T item); // перед текущим

void InsertLast(T item); // вставить последним

// удаление звеньев

void DeleteFirst(); // удалить первое звено

void DeleteCurrent(); // удалить текущее звено

void GoNext(); // сдвиг вправо текущего звена

// (=1 после применения GoNext для последнего звена списка)

void Reset(); // установить на начало списка

bool IsEnd(); // список завершен ?

T GetCurrentItem();

void SetCurrentItem(T item) { pCurrent->value = item; }

THeadList.h:

Поля:

TNode<T>* pHead; // заголовок, pFirst - звено за pHead

Методы:

THeadList();

~THeadList();

void InsertFirst(T item); // вставка звеньев после заголовка

void DeleteFirst(); // удалить первое звено

TMonom.h:

Поля:

double coef; // коэффициент монома

int index; // индекс (свертка степеней)

Методы:

TMonom()

TMonom(double coef_arg, int degX, int degY, int degZ) : coef(coef_arg)

void SetCoef(int cval)

int GetCoef(void)

void SetIndex(int ival)

int GetIndex(void)

bool operator==(const TMonom& other)

bool operator>(const TMonom& other)

bool operator<(const TMonom& other)

TPolinom.h:

Методы:

TPolinom();
TPolinom(TPolinom& other);
TPolinom(string str);
TPolinom& operator=(TPolinom& other); // присваивание
TPolinom& operator+(TPolinom& q); // сложение полиномов
// дополнительно можно реализовать:
void AddMonom(TMonom newMonom); // добавление монома
TPolinom MultMonom(TMonom monom); // умножение мономов
// TPolinom AddPolinom(TPolinom& other); // добавление полинома
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

5.Результаты:

Тесты:

1) P = 11x^2y^3z^4–7x^4y^4z^4+5xyz

   Q = 4x^2y^3z^4-8xyz+2xy

   P + Q = `15,000000x^2y^3z^4 - 7,000000x^4y^4z^4 - 3,000000xyz + 2,000000xy`

2) P = 3x^5y^2z^5–5x^4y^3z^3+7x^3y^5z

   Q = 4x^5y^2z^5-6x^4y^3z^3

   P + Q = `7,000000x^5y^2z^5 - 11,000000x^4y^3z^3 + 7,000000x^3y^5z`

3) P = 11x^2y^3z^4–7x^4y^4z^4+5xyz

   Q = 3

   P * Q = `33,000000x^2y^3z^4 - 21,000000x^4y^4z^4 + 15,000000xyz`

4) P = 4x^2y^3z^4-8xyz+2xy

   Q = 8

   P * Q = `32,000000x^2y^3z^4 - 64,000000xyz + 16,000000xy`

6.Заключение

В ходе выполнения этой лабораторной работы, я успешно справилась со всеми поставленными задачами. После изучения темы был реализован класс список на основе двухсвязного списка. 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 реализован 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:

TNode.h

```
#pragma once

template<class T>
struct TNode
{
	T value;
	TNode* pNext;
};
```

TList.h

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

template<class T>
class TList
{
protected:
	TNode<T>* pFirst; // первое звено
	TNode<T>* pCurrent; // текущее звено
	TNode<T>* pPrevious; // звено перед текущим
	TNode<T>* pLast; // последнее звено
	TNode<T>* pStop; // значение указателя, означающего конец списка
	int length; // количество звеньев в списке

public:

	TList();
	~TList();
	int GetLength() { return length; }
	bool IsEmpty(); // список пуст ?
	// вставка звеньев
	void InsertFirst(T item); // перед первым
	void InsertCurrent(T item); // перед текущим
	void InsertLast(T item);  // вставить последним

	// удаление звеньев
	void DeleteFirst(); // удалить первое звено
	void DeleteCurrent(); // удалить текущее звено

	void GoNext(); // сдвиг вправо текущего звена
	// (=1 после применения GoNext для последнего звена списка)

	void Reset(); // установить на начало списка
	bool IsEnd();  // список завершен ?

	T GetCurrentItem();
	void SetCurrentItem(T item) { pCurrent->value = item; }


};
template <class T>
TList<T>::TList()
{
	pFirst = nullptr;
	pCurrent = nullptr;
	pPrevious = nullptr;
	pLast = nullptr;
```

```
	pStop = nullptr;
	length = 0;
}

template <class T>
TList<T>::~TList()
{
	while (!IsEmpty()) {
		DeleteFirst();
	}
}

template <class T>
bool TList<T>::IsEmpty()
{
	return pFirst == nullptr;
}

template <class T>
void TList<T>::InsertFirst(T item)
{
	TNode<T>* New_Node = new TNode<T>{ item, pFirst };
	pFirst = New_Node;
	if (length == 0) { pLast = pFirst }
	length++;
}

template <class T>
void TList<T>::InsertLast(T item)
{
	TNode<T>* New_Node = new TNode<T>{ item, nullptr };
	if (IsEmpty()) { pFirst = pLast = New_Node; }
	else {
		pLast->pNext = New_Node;
		pLast = New_Node;
	}
	length++;
}

template <class T>
void TList<T>::InsertCurrent(T item)
{
	if (pCurrent == nullptr || pCurrent == pFirst) {
		InsertFirst(item);
		return;
	}
	TNode<T>* New_Node = new TNode<T>{ item, pCurrent };
	pPrevious->pNext = New_Node;
	length++;
}

template <class T>
void TList<T>::DeleteFirst()
{
	if (IsEmpty()) throw runtime_error("List is empty");
	TNode<T>* temp = pFirst;
	pFirst = pFirst->pNext;
	if (pFirst == nullptr) pLast = nullptr;
	delete temp;
	length--;
}

template <class T>
void TList<T>::DeleteCurrent()
{
```

```
	if (pCurrent == nullptr) throw runtime_error("Current node is null");
	if (pCurrent == pFirst) {
		DeleteFirst();
		return;
	}
	if (pCurrent == pLast) {
		delete pCurrent;
		pPrevious->pNext = nullptr;
		pLast = pPrevious;
		pCurrent = nullptr;
	}
	else {
		pPrevious->pNext = pCurrent->pNext;
		delete pCurrent;
		pCurrent = pPrevious->pNext;
	}
}

template <class T>
T TList<T>::GetCurrentItem()
{
	if (pCurrent == nullptr) throw runtime_error("Current node is null");
	return pCurrent->value;
}

template <class T>
void TList<T>::Reset()
{
	pCurrent = pFirst;
	pPrevious = nullptr;
}

template <class T>
void TList<T>::GoNext()
{
	pPrevious = pCurrent;
	pCurrent = pCurrent->pNext;
	if (pCurrent == nullptr || pCurrent == pStop) { pLast = pPrevious; }
}

template <class T>
bool TList<T>::IsEnd()
{
	return pCurrent == pStop;
}
```

THeadList.h

```
#pragma once
#include "TList.h"
using namespace std;


template<class T>
class THeadList : public TList<T>
{
protected:
	TNode<T>* pHead;  // заголовок, pFirst - звено за pHead
public:
	THeadList();
	~THeadList();
	void InsertFirst(T item); // вставка звеньев после заголовка
	void DeleteFirst(); // удалить первое звено
};
```

```
template<class T>
THeadList<T>::THeadList()
{
	pHead = new TNode<T>();
	if (pHead == nullptr) throw bad_alloc();
	this->pLast = pHead;
	pHead->pNext = this->pFirst;
}

template<class T>
THeadList<T>::~THeadList()
{
	delete pHead;
}

template <class T>
void THeadList<T>::InsertFirst(T item)
{
	TNode<T>* newNode = new TNode<T>{ item, nullptr };
	if (newNode == nullptr) throw bad_alloc();
	newNode->pNext = pHead->pNext;
	pHead->pNext = newNode;
	if (this->IsEmpty()) {
		this->pLast = newNode;
	}
	this->length++;
}

template <class T>
void THeadList<T>::DeleteFirst()
{
	if (this->pFirst == nullptr) throw runtime_error("List is empty");
	TNode<T>* temp = this->pFirst;
	this->pFirst = this->pFirst->pNext;
	if (this->pFirst == nullptr) {
		this->pLast = nullptr;
	}
	if (this->pCurrent == temp) {
		this->pCurrent = this->pFirst;
		this->pPrevious = nullptr;
	}
	delete temp;
	this->length--;
}
```

TMonom.h

```
#pragma once
#include<functional>
using namespace std;
struct TMonom
{
	double coef; // коэффициент монома
	int index; // индекс (свертка степеней)

	TMonom(){
		coef = 0;
		index = 0;
	}

	TMonom(double coef_arg, int degX, int degY, int degZ) : coef(coef_arg){
		if (degX < 0 || degY < 0 || degZ < 0) throw invalid_argument("Степени
должны быть неотрицательными");
		if (coef_arg == 0) throw invalid_argument("Коэффициент не может быть ра-
вен нулю");
```

```
            index = degX + degY * 10 + degZ * 100;
      }

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

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


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

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

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

TPolinom.h

```
#pragma once
#include "THeadList.h"
#include "TMonom.h"
#include <string>
using namespace std;
const int nonDisplayedZeros = 4; // Количество неотображаемых нулей при выводе коэффи-
циента полинома
                                                      // Кол-во символов после запятой =
6 - nonDisplayedZeros

const double EPSILON = 1e-6;
class TPolinom : public THeadList<TMonom>
{
public:
      TPolinom();
      TPolinom(TPolinom& other);
      TPolinom(string str);
      TPolinom& operator=(TPolinom& other); // присваивание
      TPolinom& operator+(TPolinom& q); // сложение полиномов

      // дополнительно можно реализовать:
      void AddMonom(TMonom newMonom); // добавление монома
      TPolinom MultMonom(TMonom monom); // умножение мономов
      // TPolinom AddPolinom(TPolinom& other); // добавление полинома
      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
};
```

```
TPolinom::TPolinom() :THeadList<TMonom>::THeadList(){}

TPolinom::TPolinom(TPolinom& other)
{
	pHead = new TNode<TMonom>;
	TNode<TMonom>* h = other.pHead->pNext;
	while (h != other.pStop) {
		this->AddMonom(h->value);
		h = h->pNext;
	}
}

TPolinom::TPolinom(string str)
{
	size_t pos = 0;
	while (pos < str.length()) {
		double coef = 0.0;
		int degX = 0, degY = 0, degZ = 0;
		char var;
		if (isdigit(str[pos]) || str[pos] == '-' || str[pos] == '+') {
			size_t nextPos;
			coef = stod(str.substr(pos), &nextPos);
			pos += nextPos;
		}
		while (pos < str.length() && (str[pos] == 'x' || str[pos] == 'X' ||
			str[pos] == 'y' || str[pos] == 'Y' ||
			str[pos] == 'z' || str[pos] == 'Z')) {
			var = tolower(str[pos]);
			pos++;
			if (pos < str.length() && str[pos] == '^') {
				pos++;
				size_t nextPos;
				int deg = stoi(str.substr(pos), &nextPos);
				pos += nextPos;
				switch (var) {
				case 'x': degX = deg; break;
				case 'y': degY = deg; break;
				case 'z': degZ = deg; break;
				}
			}
			else {
				switch (var) {
				case 'x': degX = 1; break;
				case 'y': degY = 1; break;
				case 'z': degZ = 1; break;
				}
			}
		}
		this->AddMonom(TMonom(coef, degX, degY, degZ));
	}
}

TPolinom& TPolinom::operator=(TPolinom& other) // присваивание
{
	if (this != &other) {
		while (!this->IsEmpty()) {
			this->DeleteFirst();
		}
		TNode<TMonom>* current = other.pHead->pNext;
		while (current != nullptr) {
			this->AddMonom(current->value);
			current = current->pNext;
		}
	}
	return *this;
```

```
}

void TPolinom::AddMonom(TMonom m) // добавление монома
{
	if (m.coef == 0) throw invalid_argument("Невозможно добавить моном с нулевым ко-
эффициентом");
	this->Reset();
	bool isAdded = false;
	while (!this->IsEnd()) {
		if (this->pCurrent->value.index == m.index) {
			this->pCurrent->value.coef += m.coef;
			if (fabs(this->pCurrent->value.coef) < EPSILON) {
				this->DeleteCurrent();
			}
			isAdded = true;
			break;
		}
		this->GoNext();
	}
	if (!isAdded) this->InsertLast(m);
	pHead->pNext = this->pFirst;
}

TPolinom TPolinom::MultMonom(TMonom monom) // умножение мономов
{
	TPolinom res(*this);
	TNode<TMonom>* current = res.pHead->pNext;
	while (current != nullptr) {
		current->value.coef *= monom.coef;
		current->value.index += monom.index;
		current = current->pNext;
	}
	return res;
}

TPolinom& TPolinom::operator+(TPolinom& other) // сложение полиномов
{
	if (other.IsEmpty()) throw invalid_argument("Не удается добавить пустой много-
член");
	TNode<TMonom>* current = other.pHead->pNext;
	while (current != nullptr) {
		this->AddMonom(current->value);
		current = current->pNext;
	}
	return *this;
}

//TPolinom TPolinom::AddPolinom(TPolinom& other)
//{
//	return *this;
//}

TPolinom TPolinom::operator*(double coef) // умножение полинома на число
{
	if (this->IsEmpty()) throw invalid_argument("Невозможно умножить пустой много-
член");
	TNode<TMonom>* current = this->pHead->pNext;
	while (current != nullptr) {
		current->value.coef *= coef;
		if (current->pNext == nullptr) break;
		current = current->pNext;
	}
	return *this;
}
```

```
//TPolinom TPolinom::operator*(TPolinom& other)
//{
//	return *this;
//}

bool TPolinom::operator==(TPolinom& other) // сравнение полиномов на равенство
{
	if (this->GetLength() != other.GetLength()) return false;
	TNode<TMonom>* thisCurrent = this->pHead->pNext;
	TNode<TMonom>* otherCurrent = other.pHead->pNext;
	while (thisCurrent != nullptr && otherCurrent != nullptr) {
		if (!(thisCurrent->value == otherCurrent->value)) { return false; }
		thisCurrent = thisCurrent->pNext;
		otherCurrent = otherCurrent->pNext;
	}
	return thisCurrent == otherCurrent;
}


string TPolinom::ToString() // перевод в строку
{
	string result;
	TNode<TMonom>* current = this->pHead->pNext;
	while (current != nullptr) {
		int degX = current->value.index % 10;
		int degY = (current->value.index / 10) % 10;
		int degZ = current->value.index / 100;
		if (!result.empty()) result += (current->value.coef > 0) ? " + " : " - ";
		else if (current->value.coef < 0) { result += "-"; }
		result += to_string(abs(current->value.coef));
		if (degX > 0) {
			result += "x";
			if (degX > 1) {
				result += "^" + to_string(degX);
			}
		}
		if (degY > 0) {
			result += "y";
			if (degY > 1) {
				result += "^" + to_string(degY);
			}
		}
		if (degZ > 0) {
			result += "z";
			if (degZ > 1) {
				result += "^" + to_string(degZ);
			}
		}
		current = current->pNext;
	}
	return result;
}
```

TPolinom_Test.cpp

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

void Menu() {
    cout << "Меню:\n";
    cout << "1. Ввести\n";
    cout << "2. Вывести\n";
    cout << "3. Добавить второй\n";
```

```
    cout << "4. Умножить на константу\n";
    cout << "5. Выход\n";
    cout << "Выверите, пожалуйста, опцию: ";
}


TPolinom ReadPolinom() {
    cout << "Введите тут: ";
    string str;
    getline(cin, str);
    return TPolinom(str);
}


int main()
{
    setlocale(LC_ALL, "Russian");
    TPolinom p1, p2, result;
    int choice;
    double coef;
    while (true) {
        Menu();
        cin >> choice;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        switch (choice) {
        case 1:
            p1 = ReadPolinom();
            break;
        case 2:
            cout << "Моном/Полином: " << p1.ToString() << endl;
            break;
        case 3:
            cout << "Введите второй  для сложения:\n";
            p2 = ReadPolinom();
            result = p1 + p2;
            cout << "Результат: " << result.ToString() << endl;
            break;
        case 4:
            cout << "Введите константу для умножения: ";
            cin >> coef;
            result = p1 * coef;
            cout << "Результат: " << result.ToString() << endl;
            break;
        case 5:
            cout << "Выход" << endl;
            return 0;
        default:
            cout << "Неправильный выбор. Пробовать снова.\n";
        }
    }
}
```

TestPolinom.cpp

```
#include "TPolinom.h"
#include <gtest.h>

TEST(Polinom, Created_polinom_is_empty) {
      TPolinom p;
      EXPECT_TRUE(p.IsEmpty());
}

TEST(Polinom, Created_polinom_with_one_monom_is_not_empty) {
      TPolinom p;
      TMonom m(1.0, 2, 3, 4);
```

```
	p.AddMonom(m);
	EXPECT_TRUE(p.IsEmpty() == false);
}

TEST(Polinom, Coefficients_add_up_if_degrees_are_equal) {
	TPolinom p;
	TMonom m1(1.5, 2, 3, 4);
	TMonom m2(2.0, 2, 3, 4);
	p.AddMonom(m1);
	p.AddMonom(m2);
	EXPECT_EQ(p.GetLength(), 1);
	EXPECT_EQ(p.GetCurrentItem().coef, 3.5);
}

TEST(Polinom, Coefficients_dont_add_up_if_degrees_are_different) {
	TPolinom polynom;
	TMonom monom1(1.0, 2, 1, 3);
	TMonom monom2(2.0, 2, 1, 3);
	polynom.AddMonom(monom1);
	polynom.AddMonom(monom2);
	EXPECT_EQ(polynom.GetLength(), 1);
	EXPECT_EQ(polynom.GetCurrentItem().coef, 3.0);
}

TEST(Polinom, Monom_delete_if_coefficient_equal_zero) {
	TPolinom polynom;
	TMonom monom1(1.0, 2, 1, 3);
	TMonom monom2(-1.0, 2, 1, 3);

	polynom.AddMonom(monom1);
	polynom.AddMonom(monom2);

	EXPECT_EQ(polynom.GetLength(), 0);
}

TEST(Polinom, Can_not_get_monom_from_empty_polinom) {
	TPolinom polynom;
	EXPECT_THROW(polynom.GetCurrentItem(), std::runtime_error);
}

TEST(Polinom, Copied_polinom_is_equal_copiable) {
	TPolinom originalPolinom;
	originalPolinom.AddMonom(TMonom(1.0, 3, 0, 0));
	TPolinom copiedPolinom(originalPolinom);
	bool flag = originalPolinom == copiedPolinom;
	EXPECT_EQ(flag, true);
}

TEST(Polinom, Assigned_polinom_is_equal_assignable) {
	TPolinom originalPolinom;
	originalPolinom.AddMonom(TMonom(1.0, 3, 0, 0));
	TPolinom assignedPolinom;
	assignedPolinom = originalPolinom;
	EXPECT_EQ(assignedPolinom.ToString(), originalPolinom.ToString());
}

TEST(Polinom, Correct_adding_operator) {
	TPolinom temp;
	temp.AddMonom(TMonom(1.0, 3, 0, 0));
	temp.AddMonom(TMonom(1.0, 3, 0, 0));
	TPolinom temp2;
	temp2.AddMonom(TMonom(2.0, 3, 0, 0));
	EXPECT_EQ(temp.ToString(), temp2.ToString());
}
```

```
TEST(Polinom, Correct_constant_multiply) {
	TPolinom temp;
	temp.AddMonom(TMonom(1.0, 3, 0, 0));
	temp = temp * 4.0;
	TPolinom temp2;
	temp2.AddMonom(TMonom(4.0, 3, 0, 0));
	EXPECT_EQ(temp.ToString(), temp2.ToString());
}
```