МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

Направление подготовки: «Программная инженерия»

Профиль подготовки: «Общий»

**Отчет по лабораторной работе**

на тему:

**«Полином»**

**Выполнил:** студент группы 3822Б1ПР2

Чернова Наталья Алексеевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись

Нижний Новгород  
2023

**Содержание**

[1. ВВЕДЕНИЕ 3](#_Toc14)

[2. ПОСТАНОВКА ЗАДАЧИ 4](#_Toc15)

[3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 5](#_Toc16)

[4. РУКОВОДСТВО ПРОГРАММИСТА 9](#_Toc17)

[4.1 Описание структуры программы 9](#_Toc18)

[4.2 Описание структур данных 9](#_Toc19)

[5. ЭКСПЕРИМЕНТЫ 13](#_Toc20)

[6. ЗАКЛЮЧЕНИЕ 14](#_Toc21)

[7.ЛИТЕРАТУРА 15](#_Toc22)

[8.ПРИЛОЖЕНИЕ 16](#_Toc23)

# 1. ВВЕДЕНИЕ

Полином – это математическое выражение, которое содержит одну или несколько переменных, а также коэффициенты и степени этих переменных.

Полином (многочлен) от n переменных – это выражение вида

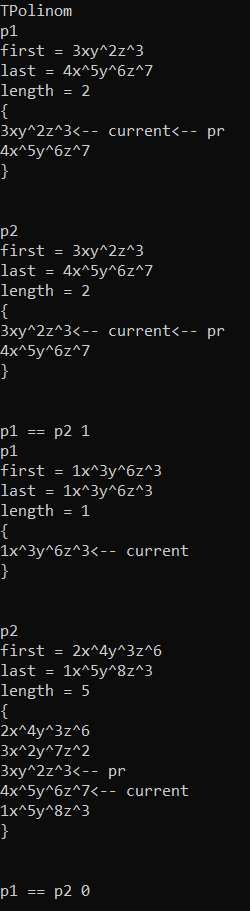
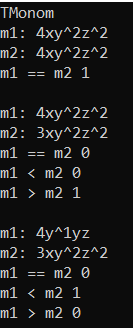
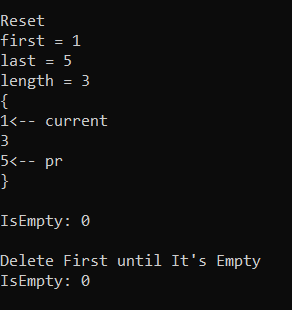
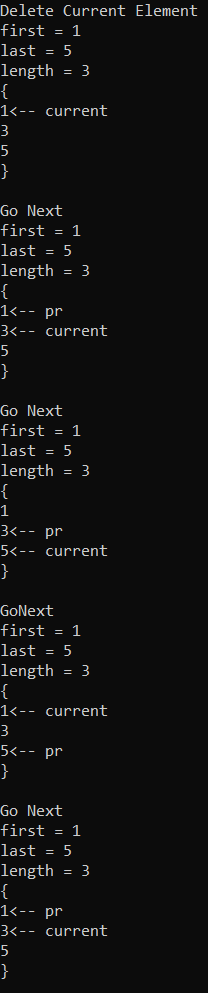
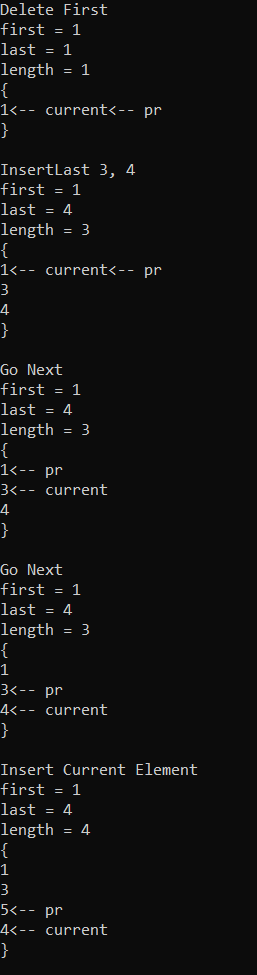
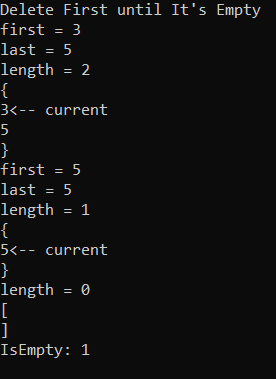
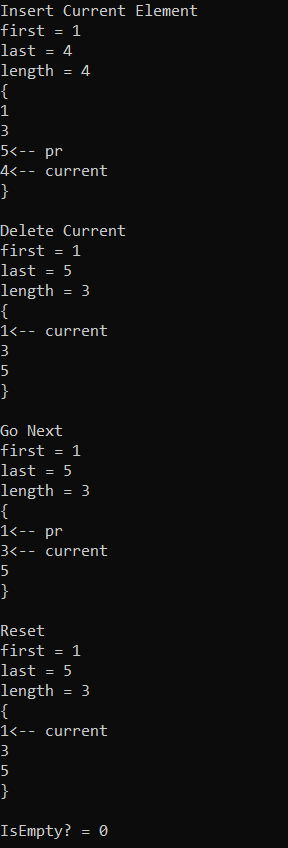
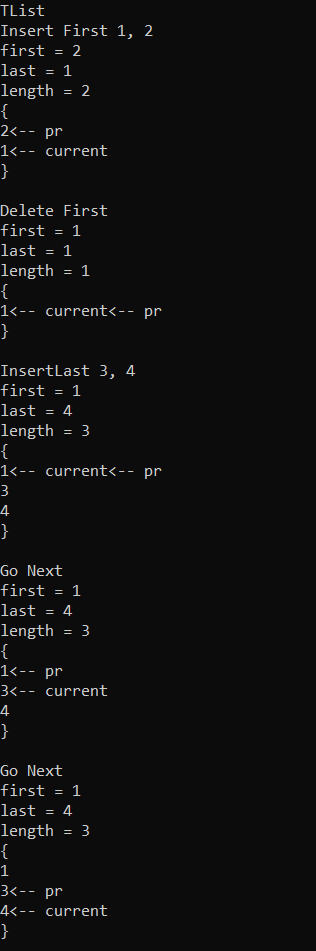
С полиномами можно проводить математические операции, такие как сложение, вычитание, умножение и деление. Язык С++ не содержит класса для представления полиномов. Поэтому в данной работе мы напишем реализацию класса полиномов.

# 2. ПОСТАНОВКА ЗАДАЧИ

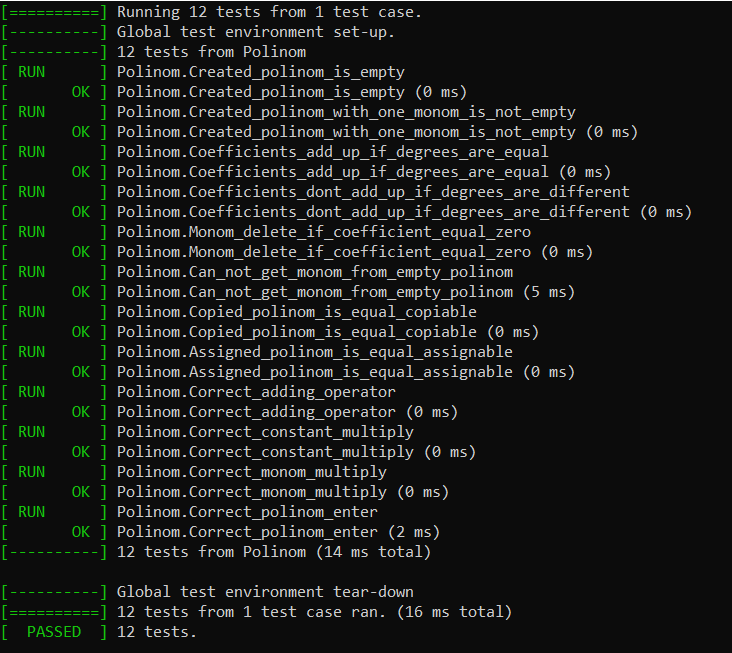
Задача данной работы – реализовать шаблонный класс полином, а также реализовать методы копирования, присваивания, сравнения, сложения и вычитания.

# 3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Проект polinom при запуске выдаёт консоль с примерами работоспособности классов TList, THeadList, TMonom и TPolinom.



А при запуске проекта polinomtest мы видим результаты работы тестов, проверяющих корректность написанных классов.



# 4. РУКОВОДСТВО ПРОГРАММИСТА

## 4.1 Описание структуры программы

Программа состоит из нескольких проектов:

* Проект polinom, содержащий файл TPolinom\_Test.cpp, где находится проверка работоспособности очереди.
* Проект listklib, содержащий файлы TNode, TList и THeadList, где реализованы вспомогательные списки.
* Проект polinomklib, содержащий файлы TMonom и TPolinom, где находится реализация классов монома и полинома.
* Проект polinomktest, содержащий тесты для проверки корректности работы класса полинома. Реализован с помощью фреймворка GoogleTest.

## 4.2 Описание структур данных

**TNode**

template<class T>

struct TNode

{

T value; //значение

TNode\* pNext; //указатель на следующий элемент

};

**TList**

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(); //конструктор

~TList(); //декструктор

int GetLength() { return length; }//установить длину

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(); //получить текущее значение

void SetCurrentItem(T item) { pCurrent->value = item; } //установить текущее значение

friend ostream& operator<<(ostream& os, const TList<T>& obj); //вывод

};

**THeadList**

template<class T>

class THeadList : public TList<T>

{

protected:

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

public:

THeadList(); //конструктор

~THeadList(); //деструктор

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

void DeleteFirst(); // удалить первое звено

void Clear(); //очистить список

void Reset(); // установить на начало списка

void GoNext(); //перейти к следующему члену

};

**TMonom**

struct TMonom

{

double coef; // коэффициент монома

int index; // индекс (свертка степеней)

TMonom() { } //конструктор

TMonom(double coef, int degX, int degY, int degZ); //конструктор

void SetCoef(int cval); //установить коэффициент

int GetCoef(void); //получить коэффициент

void SetIndex(int ival); //установить индекс

int GetIndex(void); //получить индекс

bool operator==(const TMonom& other); //оператор сравнения

bool operator>(const TMonom& other); //оператор сравнения

bool operator<(const TMonom& other); //оператор сравнения

TMonom operator=(const TMonom& other); //оператор присваивания

friend ostream& operator<<(ostream& os, const TMonom& obj); //вывод

};

**TPolinom**

class TPolinom : public THeadList<TMonom>

{

public:

TPolinom(); //конструктор

TPolinom(TPolinom& other); //конструктор копирования

TPolinom(string str); //конструктор

TPolinom& operator=(TPolinom& other); // присваивание

TPolinom& operator+(TPolinom& q); // сложение полиномов

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

bool operator==(TPolinom& other); // сравнение полиномов на равенство

string ToString(); // перевод в строку

};

# 

# 5. ЭКСПЕРИМЕНТЫ

Эксперименты проводились на ПК с следующими параметрами:

1. Операционная система: Windows 10

2. Процессор: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx, 2100 МГц

3. Версия Visual Studio: 2022

|  |  |
| --- | --- |
| Кол-во элементов (n) | Время работы оператора добавления элементов в полином (ms) |
| 1000 | 17 |
| 3000 | 157 |
| 6000 | 619 |

Здесь представлено время работы программы, добавляющей в очередь n элементов. При количестве элементов n сложность работы оператора примерно равна O(n2).

# 6. ЗАКЛЮЧЕНИЕ

В ходе работы были реализованы классы монома и полинома, классы, содержащие вспомогательные списки, а также тесты для проверки корректности работы класса полинома.

.

# 7.ЛИТЕРАТУРА

[Многочлен — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D1%87%D0%BB%D0%B5%D0%BD)

[Список (информатика) — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0))

[Pract\_ADS.pdf - Google Диск](https://drive.google.com/file/d/1aZEfnRSgA7IhMccdLTvbJ5noZlXPC1BK/view)

Лабораторный практикум. Составители:Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 105с.

# **8.ПРИЛОЖЕНИЕ**

**TNode.h**

#pragma once

template<class T>

struct TNode

{

T value;

TNode\* pNext;

};

**TList.h**

#pragma once

#include <iostream>

#include "TNode.h"

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(); //получить текущее значение

void SetCurrentItem(T item) { pCurrent->value = item; } //установить текущее значение

friend ostream& operator<<(ostream& os, const TList<T>& obj) {

//os << &obj;

if (obj.length == 0) {

os << "length = 0 \n[\n]" << endl;

return os;

}

try {

os << "first = " << obj.pFirst->value << endl;

}

catch (...) {

os << "no first" << endl;

}

try {

os << "last = " << obj.pLast->value << endl;

}

catch (...) {

os << "no last" << endl;

}

os << "length = " << obj.length << endl;

TNode<T>\* tmp = obj.pFirst;

os << "{" << endl;

while (tmp != obj.pStop)

{

os << tmp->value;// << " next" << tmp->pNext;

if (tmp == obj.pCurrent) {

os << "<-- current";

}

if (tmp == obj.pPrevious) {

os << "<-- pr";

}

os << endl;

tmp = tmp->pNext;

}

os << "}" << endl;

return os;

}

};

template <class T>

TList<T>::TList()

{

length = 0;

pStop = NULL;

pFirst = NULL;

pPrevious = NULL;

pLast = NULL;

}

template <class T>

TList<T>::~TList()

{

length = 0;

pStop = NULL;

pFirst = NULL;

pPrevious = NULL;

pLast = NULL;

}

template <class T>

bool TList<T>::IsEmpty()

{

if (length == 0) {

return true;

}

return false;

}

template <class T>

void TList<T>::InsertFirst(T item) // перед первым

{

if (length == 0) {

TNode<T>\* tnd = new TNode<T>();

tnd->value = item;

tnd->pNext = pStop;

pFirst = tnd;

pCurrent = pFirst;

pLast = pFirst;

}

else {

TNode<T>\* tnd = new TNode<T>();

tnd->value = item;

tnd->pNext = pFirst;

if (pCurrent == pFirst) {

pPrevious = tnd;

pCurrent = pFirst;

}

pFirst = tnd;

}

length++;

}

template <class T>

void TList<T>::InsertCurrent(T item) // перед текущим

{

if ((length == 0)||(pCurrent == pFirst)) {

InsertFirst(item);

return;

}

TNode<T>\* tnd = new TNode<T>();

tnd->value = item;

tnd->pNext = pCurrent;

pPrevious->pNext = tnd;

pPrevious = tnd;

length++;

}

template <class T>

void TList<T>::InsertLast(T item) // вставить последним

{

if (IsEmpty()) {

InsertFirst(item);

return;

}

TNode<T>\* tnd = new TNode<T>();

tnd->value = item;

tnd->pNext = pStop;

pLast->pNext = tnd;

pLast = tnd;

length++;

}

template <class T>

void TList<T>::DeleteFirst() // удалить первое звено

{

if (length == 0) {

throw "is Empty";

}

if (pCurrent == pFirst) {

pCurrent = pFirst->pNext;

}

if (pLast == pFirst) {

pLast = pFirst->pNext;

}

if (pPrevious == pFirst) {

pPrevious = pFirst->pNext;

}

TNode<T>\* tnd = pFirst;

pFirst = pFirst->pNext;

tnd = nullptr;

length--;

}

template <class T>

void TList<T>::DeleteCurrent() // удалить текущее звено

{

if (length == 0) {

throw "is Empty";

}

if (pCurrent == pStop) {

throw "Currend in end";

}

if ((pCurrent == pFirst) && (length == 1)) {

DeleteFirst();

length--;

return;

}

if (pCurrent->pNext == pStop) {

pLast = pPrevious;

pLast->pNext = pStop;

delete pCurrent;

Reset();

length--;

return;

}

TNode<T>\* tnd = pCurrent->pNext;

delete pCurrent;

pCurrent = tnd;

pPrevious->pNext = pCurrent;

tnd = nullptr;

length--;

}

template <class T>

T TList<T>::GetCurrentItem() //получить текущее значение

{

if (length == 0) {

throw "It's empty";

}

if (pCurrent == pStop)

throw "It's over";

return pCurrent->value;

}

template <class T>

void TList<T>::Reset() // установить на начало списка

{

pCurrent = pFirst;

pPrevious = NULL;

}

template <class T>

void TList<T>::GoNext()// сдвиг вправо текущего звена

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

{

try {

if (IsEnd()) {

throw "This is end";

}

pPrevious = pCurrent;

pCurrent = pCurrent->pNext;

}

catch (...) {}

}

template <class T>

bool TList<T>::IsEnd() // список завершен ?

{

if (IsEmpty()) {

return true;

}

if (pCurrent == pStop) {

return true;

}

if (pCurrent->pNext == pStop) {

return true;

}

return false;

}

**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

void Clear();

void Reset();

void GoNext();

};

template<class T>

THeadList<T>::THeadList():TList<T>()

{

pHead = new TNode<T>();

}

template<class T>

THeadList<T>::~THeadList()

{

delete this->pHead;

}

template <class T>

void THeadList<T>::InsertFirst(T item)

{

TList<T>::InsertFirst(item);

}

template <class T>

void THeadList<T>::DeleteFirst()

{

TList<T>::DeleteFirst();

if (length != 0) {

pHead = pFirst;

}

else {

pHead = NULL;

}

}

template<class T>

void THeadList<T>::Reset()

{

TList<T>::Reset();

pPrevious = pLast;

}

template <class T>

void THeadList<T>::Clear() {

Reset();

while (length > 0) {

DeleteFirst();

}

pCurrent = pStop;

pPrevious = pStop;

pLast = pStop;

pFirst = pStop;

length = 0;

}

template<class T>

void THeadList<T>::GoNext()

{

if (pCurrent->pNext == pStop) {

THeadList<T>::Reset();

}

else {

TList<T>::GoNext();

}

}

**TMonom.h**

#pragma once

struct TMonom

{

double coef; // коэффициент монома

int index; // индекс (свертка степеней)

TMonom() { }

TMonom(double coef, int degX, int degY, int degZ) {

this->coef = coef;

this->index = degX\*100 + degY\*10 + 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) {

if ((index == other.index) && (coef == other.coef)) {

return true;

}

return false;

}

bool operator>(const TMonom& other) {

if (index > other.index) {

return true;

}

if (index < other.index) {

return false;

}

if (coef > other.coef) {

return true;

}

return false;

}

bool operator<(const TMonom& other) {

if (index < other.index) {

return true;

}

if (index > other.index) {

return false;

}

if (coef < other.coef) {

return true;

}

return false;

}

TMonom operator=(const TMonom& other) {

if (\*this == other) {

return \*this;

}

this->coef = other.coef;

this->index = other.index;

return \*this;

}

friend ostream& operator<<(ostream& os, const TMonom& obj) {

if (&obj == nullptr) {

return os;

}

os << obj.coef;

int in = obj.index;

if ((in / 100 != 0) && (in / 100 != 1)) {

os<<"x^";

os<<std::to\_string(in/100);

}

if (in / 100 == 1) {

os << "x";

}

in %= 100;

if ((in / 10 != 0) && (in / 100 != 1)) {

os << "y^";

os << std::to\_string(in / 10);

}

if (in / 10 == 1) {

os << "y";

}

in %= 10;

if ((in != 0) && (in != 1)) {

os << "z^";

os << std::to\_string(in);

}

if (in == 1) {

os << "z";

}

return os;

}

};

**TPolinom.h**

#pragma once

#include <string>

#include "THeadList.h"

#include "TMonom.h"

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

// Кол-во символов после запятой = 6 - nonDisplayedZeros

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()

{

//а тут что-то надо?

}

TPolinom::TPolinom(TPolinom& other) :THeadList<TMonom>::THeadList()

{

TNode<TMonom>\* tnd = other.pCurrent;

TMonom tmon;

other.Reset();

while (!other.IsEnd()) {

TMonom tmon = other.pCurrent->value;

InsertLast(tmon);

other.GoNext();

}

if (!other.IsEmpty()) {

TMonom tmon = other.pCurrent->value;

InsertLast(tmon);

}

while (other.pCurrent != tnd) {

other.GoNext();

GoNext();

}

length = other.length;

tnd = nullptr;

return;

}

TPolinom& TPolinom::operator=(TPolinom& other)

{

Clear();

if (other.IsEmpty()) {

return \*this;

}

TNode<TMonom>\* st = other.pCurrent;

TMonom tmp;

other.Reset();

while (!other.IsEnd()) {

tmp = other.pCurrent->value;

InsertLast(tmp);

other.GoNext();

}

tmp = other.pCurrent->value;

InsertLast(tmp);

other.GoNext();

while (other.pCurrent != st) {

other.GoNext(); GoNext();

}

length = other.length;

st = nullptr;

return \*this;

}

void TPolinom::AddMonom(TMonom m)

{

if (m.GetCoef() == 0) return;

Reset();

TMonom\* monom = new TMonom(m);

if (!IsEmpty()) {

while ((GetCurrentItem().index < m.index) && (!IsEnd())) GoNext();

if (GetCurrentItem().index == m.index) {

pCurrent->value.coef += m.coef;

if (pCurrent->value.coef == 0) {

DeleteCurrent();

length++;

return;

}

return;

}

if (IsEnd()) {

if (GetCurrentItem().index > m.index) {

InsertFirst(m);

}

else {

InsertLast(\*monom);

return;

}

}

else InsertCurrent(\*monom);

}

else InsertFirst(\*monom);

}

TPolinom TPolinom::MultMonom(TMonom monom)

{

if (IsEmpty()) {

return \*this;

}

TPolinom tmp(\*this);

Clear();

tmp.Reset();

while (!tmp.IsEnd())

{

if (tmp.pCurrent->value == monom) {

AddMonom(monom);

}

tmp.GoNext();

}

if (tmp.pCurrent->value == monom) {

AddMonom(monom);

}

return \*this;

}

TPolinom& TPolinom::operator+(TPolinom& other)

{

if (other.IsEmpty()) {

return \*this;

}

if (this->IsEmpty()) {

return other;

}

if (length >= other.length) {

TPolinom\* tmp = new TPolinom(\*this);

other.Reset();

while (!other.IsEnd()) {

tmp->AddMonom(other.GetCurrentItem());

other.GoNext();

}

if (!other.IsEmpty()) {

tmp->AddMonom(other.GetCurrentItem());

}

return \*tmp;

}

else {

return (other + \*this);

}

return \*this;

}

TPolinom TPolinom::AddPolinom(TPolinom& other)

{

return (\*this + other);

}

bool TPolinom::operator==(TPolinom& other)

{

if (length != other.length) {

return false;

}

Reset();

other.Reset();

for (int i = 0; i < length; i++) {

TMonom a, b;

a = other.GetCurrentItem();

b = GetCurrentItem();

GoNext();

other.GoNext();

if (!(a == b)) {

return false;

}

}

return true;

}

TPolinom TPolinom::operator\*(double coef)

{

Reset();

if (IsEmpty()) {

return \*this;

}

TPolinom\* tmp = new TPolinom(\*this);

tmp->Reset();

while (!tmp->IsEnd()) {

tmp->pCurrent->value.coef \*= coef;

tmp->GoNext();

}

tmp->pCurrent->value.coef \*= coef;

return \*tmp;

}

string TPolinom::ToString()

{

string result = "";

if (IsEmpty()) return result;

Reset();

TMonom tmp;

while (!IsEnd())

{

tmp = GetCurrentItem();

result += std::to\_string(tmp.GetCoef());

int a = tmp.GetIndex();

if ((a / 100 != 0) && (a / 100 != 1)) {

result += "X^";

result += std::to\_string(a / 100);

}

if (a / 100 == 1) {

result += "X";

}

a %= 100;

if ((a / 10 != 0) && (a / 10 != 1)) {

result += "Y^";

result += std::to\_string(a / 10);

}

if (a / 10 == 1) {

result += "Y";

}

a %= 10;

if ((a != 0) && (a != 1)) {

result += "Z^";

result += std::to\_string(a);

}

if (a == 1) {

result += "Z";

}

result += ' ';

GoNext();

}

tmp = GetCurrentItem();

result += std::to\_string(tmp.GetCoef());

int a = tmp.GetIndex();

if ((a / 100 != 0) && (a / 100 != 1)) {

result += "X^";

result += std::to\_string(a / 100);

}

if (a / 100 == 1) {

result += "X";

}

a %= 100;

if ((a / 10 != 0) && (a / 10 != 1)) {

result += "Y^";

result += std::to\_string(a / 10);

}

if (a / 10 == 1) {

result += "Y";

}

a %= 10;

if ((a != 0) && (a != 1)) {

result += "Z^";

result += std::to\_string(a);

}

if (a == 1) {

result += "Z";

}

return result;

}

TPolinom::TPolinom(string str)

{

bool st = 0;

char c;

int i = 0;

TMonom m;

int\* mon\_ind = new int[3];

mon\_ind[0] = 0; mon\_ind[1] = 0; mon\_ind[2] = 0;

int w = 0;

int p = 1;

while (i < str.size()) {

c = str[i];

if (c == ' ') {

i++;

continue;

}

if ((c == '-') || (c == '+')) {

if (c == '+') {

p = 1;

}

else {

p = -1;

}

i++;

continue;

}

if ((c >= '0') && (c <= '9')) {

size\_t d = 0;

double a = stod(&str[i], &d);

if (!st) m.SetCoef(p \* a);

i += d;

continue;

}

if (c == 'X') {

if ((i + 1 < str.size()) && (str[i + 1] != 'Y')) {

w++;

}

if ((i + 1 < str.size()) && (str[i + 1] == '^')) {

i += 2;

size\_t d = 0;

int a = stoi(&str[i], &d);

i += d;

mon\_ind[0] = a;

}

else {

mon\_ind[0] = 1;

i++;

}

w++;

c = str[i];

}

if (c == 'Y') {

if ((i + 1 < str.size()) && (str[i + 1] != 'Z')) {

w++;

}

if ((i + 1 < str.size()) && (str[i + 1] == '^')) {

i += 2;

size\_t d = 0;

int a = stoi(&str[i], &d);

i += d;

mon\_ind[1] = a;

}

else { mon\_ind[1] = 1; i++; }

w++;

c = str[i];

}

if (c == 'Z') {

if ((i + 1 < str.size()) && (str[i + 1] == '^')) {

i += 2;

size\_t d = 0;

int a = stoi(&str[i], &d);

i += d;

mon\_ind[2] = a;

}

else { mon\_ind[2] = 1; i++; }

w++;

c = str[i];

}

if (w >= 3) {

m.SetIndex(mon\_ind[0] \* 100 + mon\_ind[1] \* 10 + mon\_ind[2]);

mon\_ind[0] = 0;

mon\_ind[1] = 0;

mon\_ind[2] = 0;

AddMonom(m);

m.coef = 0;

w = 0;

i++;

continue;

}

i++;

}

}

**TPolinom\_Test.cpp**

#include <iostream>

#include <ctime>

#include "TList.h"

#include "TPolinom.h"

using namespace std;

int main()

{

cout << "TList" << endl;

TList<int> t;

cout << "Insert First 1, 2" << endl;

t.InsertFirst(1);

t.InsertFirst(2);

cout << t << endl;

cout << "Delete First" << endl;

t.DeleteFirst();

cout << t << endl;

cout << "InsertLast 3, 4" << endl;

t.InsertLast(3);

t.InsertLast(4);

cout << t << endl;

cout << "Go Next" << endl;

t.GoNext();

cout << t << endl;

cout << "Go Next" << endl;

t.GoNext();

cout << t << endl;

cout << "Insert Current Element" << endl;

t.InsertCurrent(5);

cout << t << endl;

cout << "Delete Current" << endl;

t.DeleteCurrent();

cout << t << endl;

try {

cout << "Go Next" << endl;

t.GoNext();

cout << t << endl;

}

catch (...) {

cout << " Error!!! End is: " << t.IsEnd() << endl << endl;

}

cout << "Reset" << endl;

t.Reset();

cout << t << endl;

cout << "IsEmpty? = " << t.IsEmpty() << endl << endl;

cout << "Delete First until It's Empty" << endl;

while (!t.IsEmpty()) {

t.DeleteFirst();

cout << t;

}

cout << "IsEmpty: " << t.IsEmpty() << endl << endl;

cout << endl << endl << endl;

cout << "THeadList" << endl;

THeadList<int> h1;

cout << "Insert First 1, 2" << endl;

h1.InsertFirst(1);

h1.InsertFirst(2);

cout << h1 << endl;

cout << "Delete First" << endl;

h1.DeleteFirst();

cout << h1 << endl;

cout << "InsertLast 3, 4" << endl;

h1.InsertLast(3);

h1.InsertLast(4);

cout << h1 << endl;

cout << "Go Next" << endl;

h1.GoNext();

cout << h1 << endl;

cout << "Go Next" << endl;

h1.GoNext();

cout << h1 << endl;

cout << "Insert Current Element" << endl;

h1.InsertCurrent(5);

cout << h1 << endl;

cout << "Delete Current Element" << endl;

h1.DeleteCurrent();

cout << h1 << endl;

try {

cout << "Go Next" << endl;

h1.GoNext();

cout << h1 << endl;

}

catch (...) { cout << "Error!!! End is:" << h1.IsEnd() << endl << endl; }

try {

cout << "Go Next" << endl;

h1.GoNext();

cout << h1 << endl;

}

catch (...) { cout << "Error!!! End is:" << h1.IsEnd() << endl << endl; }

try {

cout << "GoNext" << endl;

h1.GoNext();

cout << h1 << endl;

}

catch (...) { cout << "Error!!! End is:" << h1.IsEnd() << endl << endl; }

try {

cout << "Go Next" << endl;

h1.GoNext();

cout << h1 << endl;

}

catch (...) { cout << "Error!!! End is:" << h1.IsEnd() << endl << endl; }

cout << "Reset" << endl;

h1.Reset();

cout << h1 << endl;

cout << "IsEmpty: " << h1.IsEmpty() << endl << endl;

cout << "Delete First until It's Empty" << endl;

while (!t.IsEmpty()) {

h1.DeleteFirst();

cout << h1;

}

cout << "IsEmpty: " << h1.IsEmpty() << endl << endl;

cout << endl << endl << endl;

cout << "TMonom" << endl;

TMonom m1(4, 1, 2, 2);

TMonom m2(4, 1, 2, 2);

cout << "m1: " << m1 << endl;

cout << "m2: " << m2 << endl;

cout << "m1 == m2 " << (m1 == m2) << endl << endl;

m2.SetCoef(3);

cout << "m1: " << m1 << endl;

cout << "m2: " << m2 << endl;

cout << "m1 == m2 " << (m1 == m2) << endl;

cout << "m1 < m2 " << (m1 < m2) << endl;

cout << "m1 > m2 " << (m1 > m2) << endl << endl;

m1.SetIndex(11);

cout << "m1: " << m1 << endl;

cout << "m2: " << m2 << endl;

cout << "m1 == m2 " << (m1 == m2) << endl;

cout << "m1 < m2 " << (m1 < m2) << endl;

cout << "m1 > m2 " << (m1 > m2) << endl << endl;

cout << endl << endl << endl;

cout << "TPolinom" << endl;

TPolinom p1;

p1.AddMonom(TMonom(3, 1, 2, 3));

p1.AddMonom(TMonom(4, 5, 6, 7));

TPolinom p2;

p2.AddMonom(TMonom(3, 1, 2, 3));

p2.AddMonom(TMonom(4, 5, 6, 7));

cout << "p1" << endl << p1 << endl << endl;

cout << "p2" << endl << p2 << endl << endl;

cout << "p1 == p2 " << (p1 == p2) << endl;

p2.AddMonom(TMonom(3, 2, 7, 2));

p2.AddMonom(TMonom(2, 4, 3, 6));

p2.AddMonom(TMonom(1, 5, 8, 3));

p1.Clear();

p1.AddMonom(TMonom(1, 3, 6, 3));

cout << "p1" << endl << p1 << endl << endl;

cout << "p2" << endl << p2 << endl << endl;

cout << "p1 == p2 " << (p1 == p2) << endl;

/\*

unsigned int start\_time = clock();

int n = 6000;

TPolinom p;

for (int i = 0; i < n; i++) {

p.AddMonom(TMonom(i, i, i, i));

}

//cout << p << endl;

unsigned int end\_time = clock();

unsigned int search\_time = end\_time - start\_time;

cout << search\_time;

\*/

}

**TestPolinom.cpp**

#include <gtest.h>

#include "TPolinom.h"

TEST(Polinom, Created\_polinom\_is\_empty) {

TPolinom polinom;

EXPECT\_EQ(true, polinom.IsEmpty());

}

TEST(Polinom, Created\_polinom\_with\_one\_monom\_is\_not\_empty) {

TPolinom polinom;

polinom.AddMonom(TMonom(1, 1, 1, 1));

EXPECT\_EQ(false, polinom.IsEmpty());

}

TEST(Polinom, Coefficients\_add\_up\_if\_degrees\_are\_equal) {

TPolinom polinom;

polinom.AddMonom(TMonom(1, 1, 1, 1));

polinom.AddMonom(TMonom(1, 1, 1, 1));

EXPECT\_EQ(2, polinom.GetCurrentItem().GetCoef());

}

TEST(Polinom, Coefficients\_dont\_add\_up\_if\_degrees\_are\_different) {

TPolinom polinom;

polinom.AddMonom(TMonom(1, 1, 1, 1));

polinom.AddMonom(TMonom(1, 2, 3, 4));

EXPECT\_EQ(1, polinom.GetCurrentItem().GetCoef());

}

TEST(Polinom, Monom\_delete\_if\_coefficient\_equal\_zero) {

TPolinom polinom;

polinom.AddMonom(TMonom(1, 1, 1, 1));

polinom.AddMonom(TMonom(-1, 1, 1, 1));

EXPECT\_EQ(1, polinom.IsEmpty());

}

TEST(Polinom, Can\_not\_get\_monom\_from\_empty\_polinom) {

TPolinom polinom;

ASSERT\_ANY\_THROW(polinom.GetCurrentItem());

}

TEST(Polinom, Copied\_polinom\_is\_equal\_copiable) {

TPolinom polinom;

polinom.AddMonom(TMonom(1, 2, 3, 4));

TPolinom polinom2(polinom);

bool ans = (polinom2 == polinom);

EXPECT\_EQ(ans, true);

}

TEST(Polinom, Assigned\_polinom\_is\_equal\_assignable) {

TPolinom polinom;

polinom.AddMonom(TMonom(1, 2, 3, 4));

TPolinom polinom2 = polinom;

bool ans = (polinom2 == polinom);

EXPECT\_EQ(ans, true);

}

TEST(Polinom, Correct\_adding\_operator) {

TPolinom polinom;

TMonom m2(1, 5, 3, 4);

TMonom m1(1, 2, 3, 4);

polinom.AddMonom(m2);

polinom.AddMonom(m1);

polinom.Reset();

bool ans = (m1.GetCoef() == polinom.GetCurrentItem().GetCoef());

ans = ans && (m1.GetIndex() == polinom.GetCurrentItem().GetIndex());

polinom.GoNext();

ans = ans && (m2.GetCoef() == polinom.GetCurrentItem().GetCoef());

ans = ans && (m2.GetIndex() == polinom.GetCurrentItem().GetIndex());

EXPECT\_EQ(true, ans);

}

TEST(Polinom, Correct\_constant\_multiply) {

TPolinom polinom;

polinom.AddMonom(TMonom(3, 1, 2, 3));

polinom.AddMonom(TMonom(2, 1, 4, 3));

TPolinom polinom2 = polinom \* 3;

TPolinom polinom3;

polinom3.AddMonom(TMonom(9, 1, 2, 3));

polinom3.AddMonom(TMonom(6, 1, 4, 3));

bool ans = (polinom2 == polinom3);

EXPECT\_EQ(true, ans);

}

TEST(Polinom, Correct\_monom\_multiply) {

TPolinom polinom;

polinom.AddMonom(TMonom(3, 1, 2, 3));

polinom.AddMonom(TMonom(2, 1, 4, 3));

polinom.AddMonom(TMonom(1, 1, 6, 3));

polinom.MultMonom(TMonom(3, 1, 2, 3));

bool ans = (polinom.GetCurrentItem() == TMonom(3, 1, 2, 3));

EXPECT\_EQ(true, ans);

}

TEST(Polinom, Correct\_polinom\_enter) {

TPolinom p5("4X^9YZ - 5XZ");

p5.Reset();

TMonom m = p5.GetCurrentItem();

p5.GoNext();

TMonom m2 = p5.GetCurrentItem();

bool ans = ((m2.coef == 4) && (m2.index == 911));

ans = ans && ((m.coef == -5) && (m.index == 101));

EXPECT\_EQ(true, ans);

}