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: «Фундаментальная информатика и информационные технологии»

**ОТЧЕТ**

по лабораторной работе

**ДИНАМИЧЕСКАЯ СТРУКТУРА ХРАНЕНИЯ ДАННЫХ «СПИСОК». РЕАЛИЗАЦИЯ ВЫЧИСЛЕНИЙ ПРИ РАБОТЕ С ПОЛИНОМАМИ.**

**Выполнил:** студент группы

382006-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.Н. Грачёв

Подпись

**Проверил:** к.ф.-м.н., доц.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К.А. Баркалов

Подпись

Нижний Новгород  
2022

Оглавление

[Описание алгоритма работы динамической структуры данных «Список». 3](#_Toc99437490)

[Общая структура 4](#_Toc99437491)

[Конструктор 4](#_Toc99437492)

[Добавление элемента в список 4](#_Toc99437493)

[Удаление элементов 5](#_Toc99437494)

[Получение и изменение текущего элемента 5](#_Toc99437495)

[Итератор и обход списка 5](#_Toc99437496)

[Модифицированный список. 6](#_Toc99437497)

[Структура данных для работы с полиномами 6](#_Toc99437498)

[Структура монома 7](#_Toc99437499)

[Конструктор класса TPolinom 7](#_Toc99437500)

[Конструктор копирования TPolinom 7](#_Toc99437501)

[Операция присваивания полиномов 7](#_Toc99437502)

[Сложение полинома и монома 8](#_Toc99437503)

[Сложение двух полиномов 8](#_Toc99437504)

[Умножение полинома на коэффициент 8](#_Toc99437505)

[Умножение двух полиномов 9](#_Toc99437506)

[Сравнение полиномов на равенство 9](#_Toc99437507)

[4. Описание программы 10](#_Toc99437508)

[Тестирующий блок «TPolinom\_Test.cpp» 10](#_Toc99437509)

[Тестирующий блок Google Test «TestPolinom.cpp» 11](#_Toc99437510)

[5. Результаты 13](#_Toc99437511)

[6. Заключение 13](#_Toc99437512)

[7. Литература 13](#_Toc99437513)

[8. Приложение 14](#_Toc99437514)

[TList.h 14](#_Toc99437515)

[THeadList.h 18](#_Toc99437516)

[TMonom.h 19](#_Toc99437517)

[TPolinom.h 20](#_Toc99437518)

[TPolinom\_Test.cpp 25](#_Toc99437519)

[GoogleTest – TestPolinom.cpp 27](#_Toc99437520)

# Введение

В своей работе программист часто сталкивается с ситуацией, когда необходимо иметь массив элементов, размер которого заранее не известен. Необходимо изменять размер «на лету». Хорошим решением такой задачи выступает динамическая структура хранения данных «список» - память выделяется и освобождается по мере поступления и удаления элементов.

Динамический список может стать оптимальной структурой хранения полиномов. Как мы знаем, полиномы состоят из мономов – отдельных слагаемых. Мономы будут являться элементами списка, а он, в свою очередь, полиномом. Таким образом, можно реализовать на основе динамических списков удобный класс для работы с полиномами.

# Постановка задачи

Необходимо разработать 2 класса – динамическую структуру хранения данных «Список» (TList) – будет являться шаблонным классом, позволяющий добавлять, удалять, изменять и получать элементы списка, предполагающий наличие итератора для передвижения по списку; и класс для работы с полиномами, основанный на списках (TPolinom), с операциями сложения, вычитания, умножения. Разработанные классы протестировать.

# Описание алгоритмов

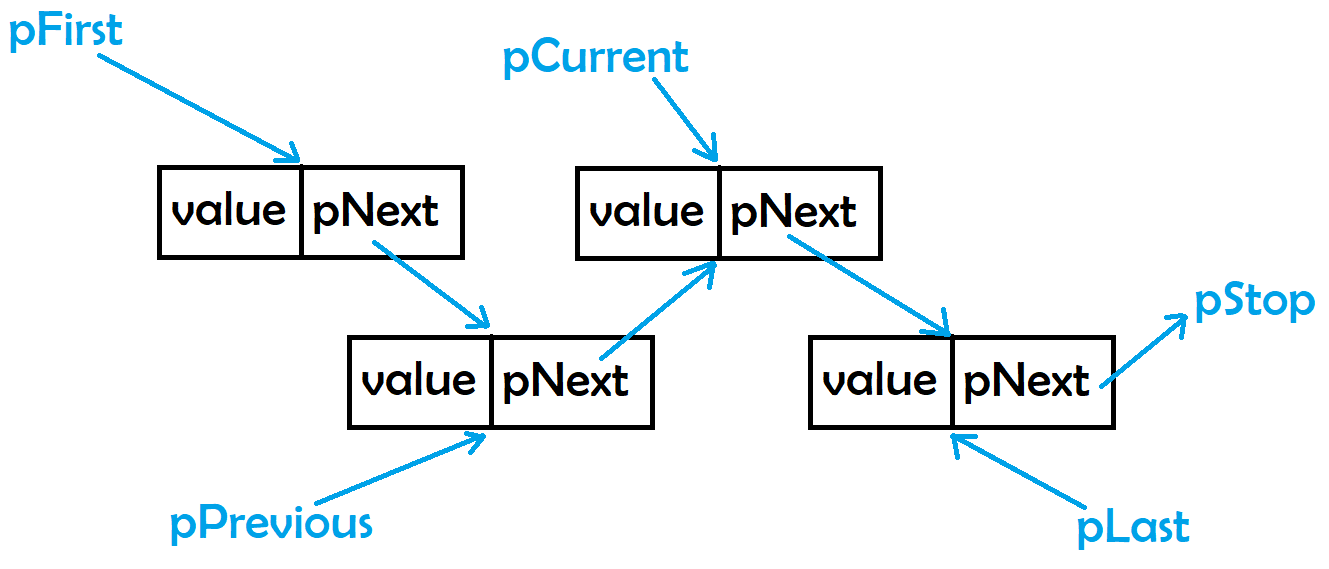
## Описание алгоритма работы динамической структуры данных «Список».

Данная динамическая структура предполагает хранение любых элементов и реализует следующий функционал:

* Итератор – указатель на текущий элемент в списке, который можно перемещать на следующий и ставить на начало, возможность обхода всего списка.
* Добавить элемент в список: в начало, в конец, перед текущим элементом,
* Удаление элемента: удалить первый, удалить текущий,
* Получить/изменить текущий элемент,
* Узнать пуст ли список, достиг ли итератор конца списка.

### Общая структура

Список состоит из добавленных в него узлов. Узел содержит в себе значение элемента и указатель на следующий узел. Также для корректной работы списка необходимо знать адрес первого узла - pFirst, адрес последнего узла – pLast, текущую длину. Для итерации понадобятся дополнительные указатели: на текущий узел – pCurrent, на предыдущий узел – pPrevious. Имеет смысл создать нулевой указатель pStop – остальным указателям, там где необходимо присвоить нулевое значение, мы будем присваивать pStop – это понадобится для дальнейшей модификации класса TList.

Таким образом имеем следующую структуру:

### Конструктор

После создания список должен быть пустым: pStop присваиваем nullptr, всем остальным присваиваем значение pStop, длина равна 0.

### Добавление элемента в список

Элемент можно добавить в начало, перед текущим и в конец.

**Добавление в начало:** если в списке уже имелись элементы, то необходимо новому элементу указатель pNext присвоить pFirst, а затем pFirst переместить на новый элемент. Если же это до добавления список был пуст, то необходимо будет еще и pLast перевести на новый элемент. Увеличить счетчик.

**Добавление перед текущим:** Если текущий элемент – первый, то делаем добавление в начало. Если нет, то элементу, находящимся перед текущим, pNext присваиваем указатель на новый элемент, а pNext у нового – на pCurrent. Увеличить счетчик.

**Добавление в конец:** если список до добавления был пуст, то делаем добавление в начало. Иначе мы должны pNext нового элемента присвоить pStop, а указатель pNext элемента на pLast перевести на новый элемент, затем pLast перевести на новый элемент. Увеличить счетчик.

### Удаление элементов

Удалить можно первый и текущий элемент.

**Удаление первого:** разрешается только если список не пуст. Необходимо запомнить элемент по указателю pFirst – пусть это будет pTemp, затем pFirst передвинуть на следующий элемент, а в конце удалить элемент по указателю pTemp. Уменьшить счетчик.

**Удаление текущего:** разрешается удалять элемент, только если указатель pCurrent не равен pStop. Если текущий элемент – это первый, то производим удаление первого элемента. Иначе, мы запоминаем элемент по указателю pCurrent (создаем указатель на него – pTemp), затем двигаем указатель pCurrent на следующий элемент, затем pNext элемента по указателю pPrevious присваиваем сдвинутый pCurrent.

### Получение и изменение текущего элемента

Чтобы получить текущий элемент, мы должны вернуть значение value по указателю pCurrent.

Чтобы изменить значение текущего элемента, необходимо элементу по указателю pCurrent присвоить полученное значение.

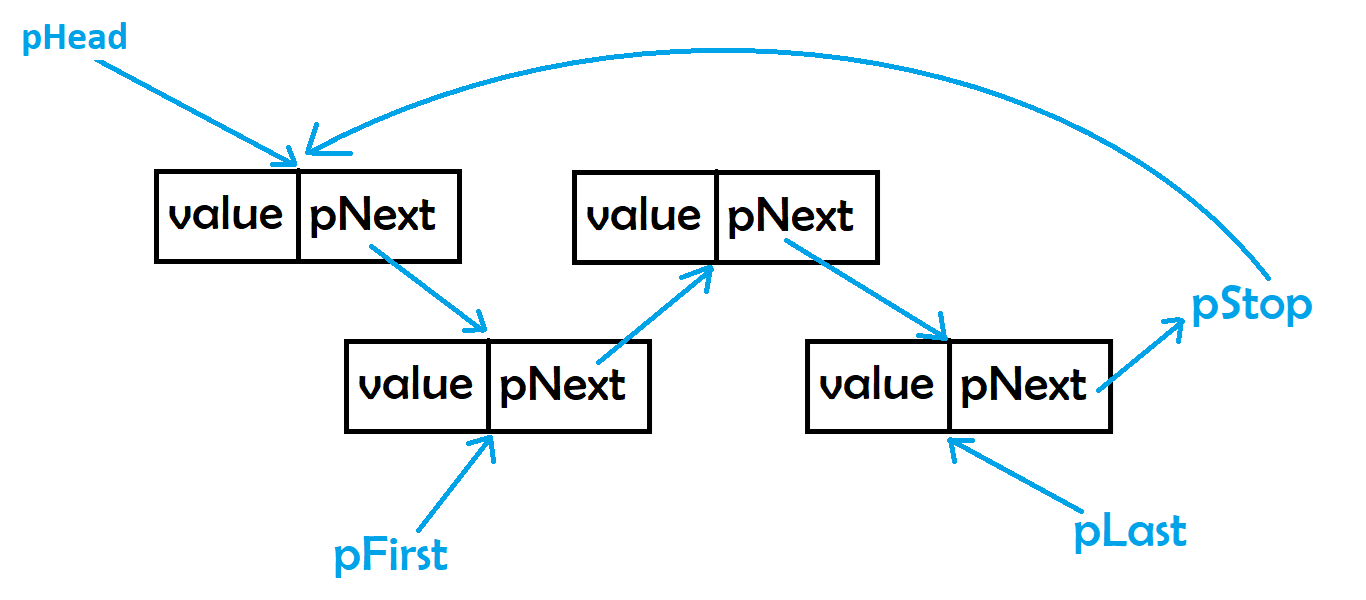
### Итератор и обход списка

Указатель на текущий элемент pCurrent можно двигать на следующий элемент, путем присваивания указателей pCurrent и pPrevious соответствующие указатели pNext. Для возврата в начало списка необходимо перевести pCurrent на pFirst, а pPrevious на pStop.

Для удобной итерации по списку имеет смысл написать функцию, которая возвращает true, если pCurrent достиг pStop (текущий элемент достиг конца списка) и false в противном случае. При помощи этой функции и перемещения текущего элемента на следующий элемент можно реализовать полный обход списка.

## Модифицированный список.

Для написания алгоритмов работы с полиномами, будет удобно реализовать список с барьером следующего вида:



При создании такого списка, мы всегда реализуем первый элемент по указателю pHead – голову списка, а также pStop присваиваем pHead, чтобы когда указатель на текущий элемент достигал конца списка. Он переводился на элемент по указателю pHead.

Унаследовав весь функционал от обыкновенного списка, понадобится изменить лишь операции удаления и добавления первого элемента – необходимо будет после завершения операции устанавливать указатель на следующий элемент у pHead на новый pFirst.

## Структура данных для работы с полиномами

Класс «TPolinom» наследуется от модифицированного списка, хранящего мономы. Он должен реализовывать следующие методы:

* Конструктор копирования, оператор присваивания,
* Сложение полинома и монома,
* Сложение полиномов,
* Умножение полиномов,
* Умножение полинома на коэффициент,
* Сравнение двух полиномов на равенство.

### Структура монома

Так как полином – это сумма конечного количества мономов, то имеет смысл написать структуру «TMonom» - из таких объектов и будет состоять наш список, который, в свою очередь, представляет собой полином. Моном сделаем зависимым от 3-х переменных: x, y, z. У каждой переменной есть ее степень, а у всего монома имеется коэффициент. Поэтому в структуре достаточно завести 4 поля – коэффициент, степень x, степень y, степень z.

В списке мономы необходимо хранить упорядоченно, поэтому для мономов нужно реализовать операцию сравнения на больше/меньше и равенство. Под упорядоченным хранением имеется ввиду хранение в лексикографическом порядке. Сравнивать на равенство следует лишь степени полинома, игнорируя коэффициент – это необходимо для правильного алгоритма работы класса TPolinom. Если же нужно сравнить мономы на точное совпадение, тогда необходимо реализовать дополнительный метод сравнения.

### Конструктор класса TPolinom

Так как полином наследуется от модифицированного списка, то необходимо вызвать конструктор списка. При создании полинома он равен 0, но у нас есть самый первый элемент по указателю pHead, его степени необходимо проинициализировать: deg(x) = 0, deg(y) = 0, deg(z) = -1. Делается это для того, чтобы при добавлении нового монома, такого, что он должен будет разместиться в конце списка, его можно было сравнить со степенью z=-1 – моном, который находится по указателю pHead всегда будет меньше и потому произойдет корректное добавление нового монома.

### Конструктор копирования TPolinom

Чтобы скопировать один полином в другой, необходимо в таком же порядке разместить мономы. Для этого циклом пройдем список копируемого полинома от начала до конца и на каждой итерации будем вставлять моном в конец списка нового полинома.

### Операция присваивания полиномов

Для того, чтобы присвоить один полином другому, необходимо очистить список того полинома, которому будет присваиваться значение, через цикл «пока список не пусть – удаляем первый элемент». Далее можно сделать то же самое, что и при копировании полинома.

### Сложение полинома и монома

Так как имеется правило хранения полинома– список упорядочен от наибольших к наименьшим мономам, то при сложении полинома и монома необходимо сделать следующее:  
Перемещаем итератор по списку и сравниваем мономы: если моном, который мы хотим добавить, меньше – тогда перемещаем итератор на следующий элемент. Если мономы равны – тогда необходимо сложить коэффициенты, но следует учитывать то, что после сложения коэффициент может стать равным 0, а в таком случае необходимо удалить данный элемент из списка, чтобы не хранить мономы с нулевыми коэффициентами. Иначе, если мы встретили в списке моном, который больше добавляемого, тогда необходимо вставить новый моном перед текущим – такая функция уже реализована в списке.

### Сложение двух полиномов

Чтобы сложить два полинома, необходимо завести временную переменную под результат сложения. В результат сразу скопируем один из полиномов. Далее с помощью итератора обойдем списки обоих полиномов, для удобства обозначим результат, как (1), а другой полином как (2):  
Если моном из (1) больше, чем текущий моном из (2), то двигаем итератор у (1) дальше; если меньше – тогда делаем вставку монома из (2) перед текущим мономом из (1) и двигаем итератор (2). Если же мономы оказались равны, то необходимо сложить коэффициенты (если коэффициент равен 0, то нужно удалить из (1) текущий моном), а затем у (1) и (2) переместить итератор дальше. После того, как был завершен обход обоих списков, необходимо вернуть результат.

### Умножение полинома на коэффициент

Чтобы умножить полином на коэффициент, необходимо при помощи итератора обойти весь список полинома и коэффициент каждого монома умножить на нужное значение.

### Умножение двух полиномов

Необходимо подготовить вспомогательную функцию – умножение полинома на моном. Чтобы выполнить это, нужно обойти весь список полинома и коэффициент каждого монома умножить на коэффициент полученного через параметр монома, а соответствующие степени сложить.

Имея уже готовые функции сложения полиномов и умножения полинома на моном, реализовать умножение полиномов не составит труда. Для этого необходимо создать временную переменную под результат, а затем сделать проход списка одного из полиномов: на каждой итерации необходимо в результат добавлять умноженный полином на моном итерируемого полинома. После окончания работы цикла, необходимо вернуть результат.

### Сравнение полиномов на равенство

Для этого необходимо в первую очередь проверить длину списков обоих полиномов – если они не равны, то полиномы не равны. Иначе, нам необходимо одновременно итерироваться по обоим спискам и сравнивать на точное совпадение соответствующих мономов. Если какой-либо из условий не было выполнено, то полиномы не равны.

# Описание программы

Программа содержит 4 основных файла с реализацией алгоритмов:

* «TList.h» - содержит реализацию динамической структуры данных «Список».
* «THeadList.h» - содержит реализацию модифицированного списка.
* «TMonom.h» - содержит реализацию структуры монома и функции их сравнения.
* «TPolinom.h» - содержит реализацию структуры хранения полиномов и их арифметических операций.

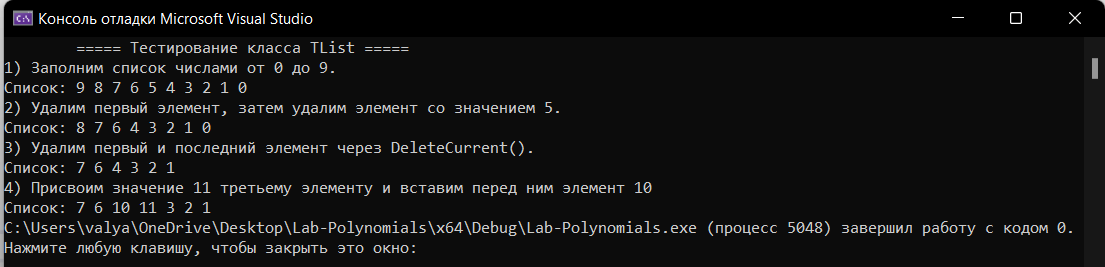
Также имеется 2 тестирующих блока:

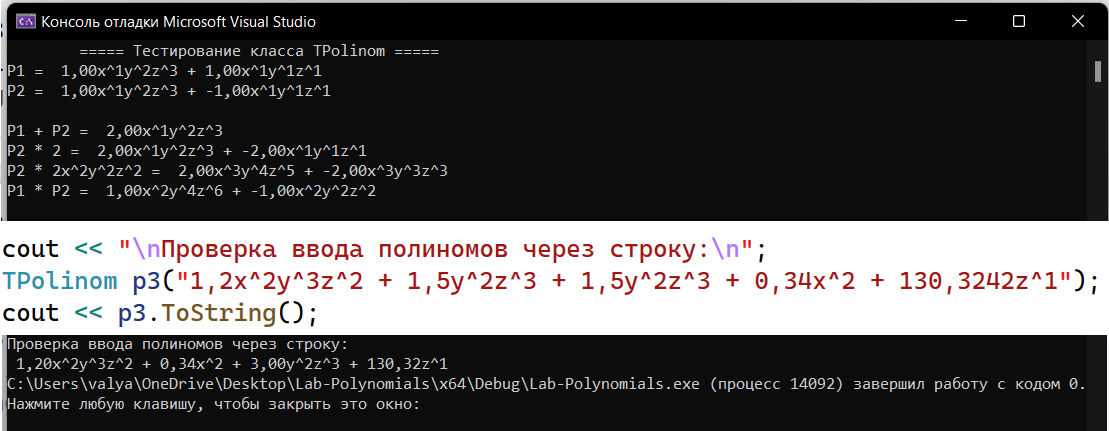
* «TPolinom\_Test.cpp» - тестирование основных функций класса «TPolinom» с выводом результатов на консоль.
* Google Test – «TestPolinom» - тестирование всего функционала класса TPolinom при помощи средств тестирования Google Test.

Код всех классов и структур приведен в разделе «Приложение».

## Тестирующий блок «TPolinom\_Test.cpp»

Содержит в себе 2 тестирующих функции – TList\_Test(), TPolinom\_Test().

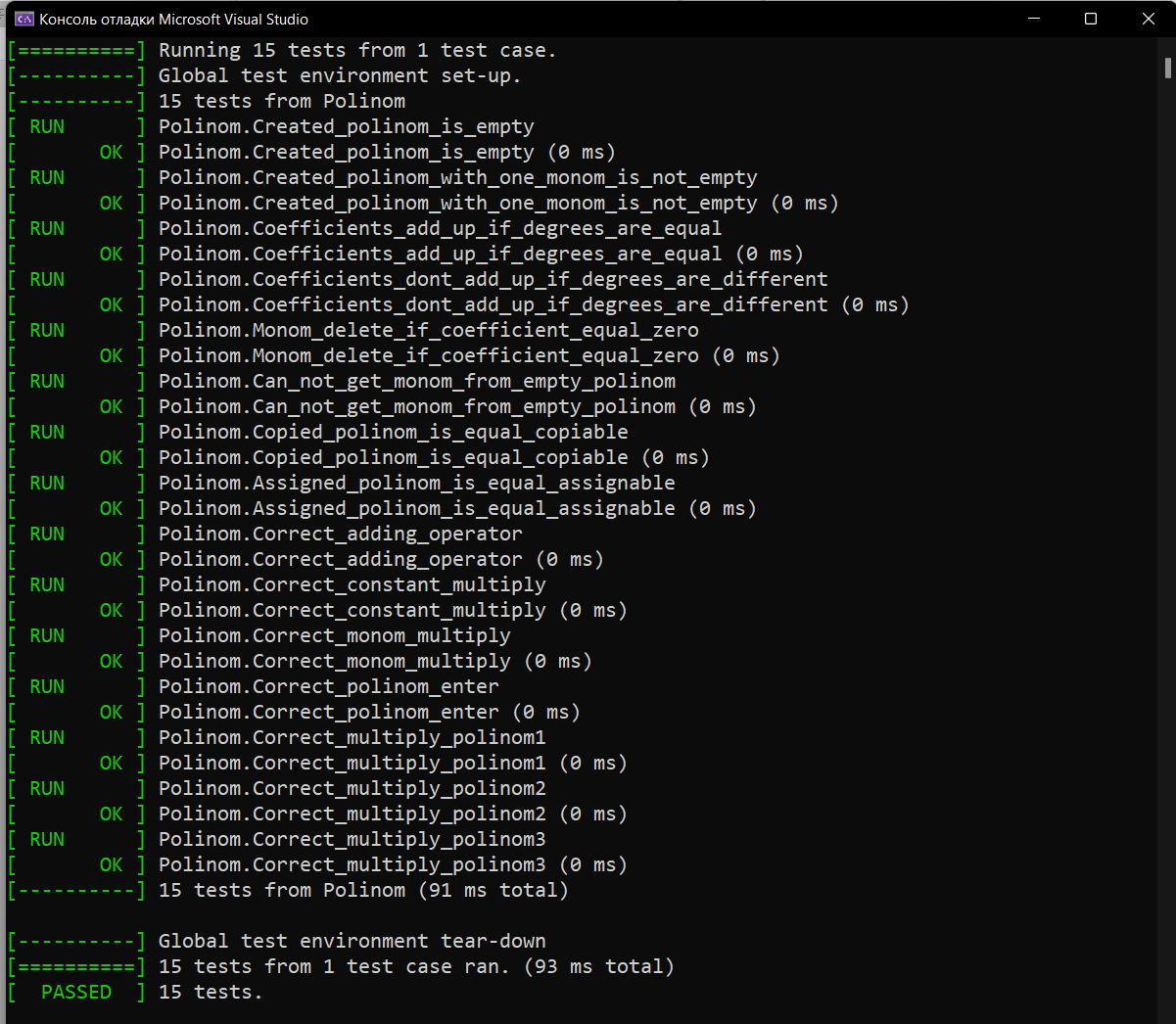
Функция **TList\_Test()** тестирует работу односвязного списка следующим образом:

Функция **TPolinom\_Test()** тестирует работу полиномов следующим образом:

## Тестирующий блок Google Test «TestPolinom.cpp»

В Google Test проводятся следующие тесты над полиномами:

* Созданный полином равен 0,
* Список полинома с одним мономом не пуст,
* Коэффициенты мономов складываются если их степени равны,
* Коэффициенты не складываются если степени мономов различны,
* Моном удаляется из полинома, если его коэффициент равен 0,
* Не можем получить моном из полинома, равного 0,
* Скопированный полином равен копируемому,
* Присвоенный полином равен присвоенному,
* Оператор сложения дает верные результаты,
* Умножение полинома на константу дает верные результаты,
* Умножение двух полиномов дает верные результаты,
* Ввод полиномов со строки работает корректно.

Все тесты успешно пройдены:

# Результаты

Были разработаны 2 класса, один из которых реализует работу с односвязными списками, а другой на их основе реализует работу с полиномами, арифметические операции над ними. Был успешно протестирован функционал обоих классов.

# Заключение

Таким образом, мы увидели как работает динамическая структура хранения данных «Список» и смогли найти ей практическое применение в виде разработки класса «Полином». Написанные классы можно эффективно применять при решении математических задач.

# Литература

1. Microsoft [электронный ресурс]: // Документация по языку C++

URL: <https://docs.microsoft.com/ru-ru/cpp/cpp/>

1. Р. Лафоре Объектно-ориентированное программирование в C++ – Санкт-Петербург: “Питер”, 2018
2. <https://ru.wikipedia.org/wiki/Связный_список>
3. https://ru.wikipedia.org/wiki/Многочлен

***Ссылка на GitHub на репозиторий с проектом:***<https://github.com/TechnoWolf96/Lab-Polynomials.git>

# Приложение

## TList.h

#pragma once

#include <iostream>

using namespace std;

template<class T>

struct TNode

{

T value;

TNode\* pNext;

};

template<class T>

class TList

{

protected:

TNode<T>\* pFirst, \* pCurrent, \* pPrevious, \* pLast, \* pStop;

int length;

public:

TList();

~TList();

int GetLength() { return length; }

bool IsEmpty();

void InsertFirst(T item);

void InsertCurrent(T item);

void InsertLast(T item);

void DeleteFirst();

void DeleteCurrent();

void GoNext();

void Reset();

bool IsEnd();

T GetCurrentItem();

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

};

template <class T>

TList<T>::TList()

{

pStop = nullptr;

pFirst = pLast = pPrevious = pCurrent = pStop;

length = 0;

}

template <class T>

TList<T>::~TList()

{

while (pFirst != pStop)

{

TNode<T>\* tmp = pFirst;

pFirst = pFirst->pNext;

delete tmp;

}

}

template <class T>

bool TList<T>::IsEmpty()

{

return pFirst == pStop;

}

template <class T>

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

{

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

newNode->value = item;

newNode->pNext = pFirst;

pFirst = newNode;

length++;

if (length == 1)

pLast = pFirst;

}

template <class T>

void TList<T>::InsertLast(T item)

{

if (length > 0)

{

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

newNode->value = item;

newNode->pNext = pStop;

pLast->pNext = newNode;

pLast = newNode;

length++;

}

else

{

InsertFirst(item);

}

}

template <class T>

void TList<T>::InsertCurrent(T item)

{

if (pCurrent == pFirst)

{

InsertFirst(item);

pPrevious = pFirst;

}

else if (pPrevious == pLast)

{

InsertLast(item);

pPrevious = pLast;

}

else

{

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

newNode->value = item;

newNode->pNext = pCurrent;

pPrevious->pNext = newNode;

pCurrent = newNode;

length++;

}

}

template <class T>

void TList<T>::DeleteFirst()

{

if (pFirst == pStop)

throw "Can't delete element from empty list";

TNode<T>\* tmp = pFirst;

pFirst = pFirst->pNext;

delete tmp;

length--;

}

template <class T>

void TList<T>::DeleteCurrent()

{

if (pCurrent == pStop)

throw "Can't delete element from empty list";

if (pFirst == pCurrent)

{

DeleteFirst();

pPrevious = pStop; // After delete first element pPrevious was null

pCurrent = pFirst;

}

else

{

TNode<T>\* tmp = pCurrent;

pCurrent = pCurrent->pNext;

pPrevious->pNext = pCurrent;

delete tmp;

length--;

}

}

template <class T>

T TList<T>::GetCurrentItem()

{

if (pCurrent == pStop)

throw "List is end";

return pCurrent->value;

}

template <class T>

void TList<T>::Reset()

{

pPrevious = pStop;

pCurrent = pFirst;

}

template <class T>

void TList<T>::GoNext()

{

pPrevious = pCurrent;

pCurrent = pCurrent->pNext;

}

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;

public:

THeadList();

~THeadList();

void InsertFirst(T item);

void DeleteFirst();

};

template<class T>

THeadList<T>::THeadList()

{

pHead = new TNode<T>();

pHead->pNext = pHead;

pFirst = pLast = pPrevious = pCurrent = pStop = pHead;

length = 0;

}

template<class T>

THeadList<T>::~THeadList()

{

TList<T>::~TList();

delete pHead;

}

template <class T>

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

{

TList<T>::InsertFirst(item);

pHead->pNext = pFirst;

}

template <class T>

void THeadList<T>::DeleteFirst()

{

TList<T>::DeleteFirst();

pHead->pNext = pFirst;

}

## TMonom.h

#pragma once

struct TMonom

{

double coef;

int degX, degY, degZ;

TMonom()

{

this->coef = 0;

this->degX = 0;

this->degY = 0;

this->degZ = 0;

}

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

{

this->coef = coef;

this->degX = degX;

this->degY = degY;

this->degZ = degZ;

}

bool operator==(const TMonom& other)

{

return degX == other.degX && degY == other.degY && degZ == other.degZ;

}

bool operator>(const TMonom& other)

{

if (degX > other.degX) return true;

else if (degX == other.degX && degY > other.degY) return true;

else if (degX == other.degX && degY == other.degY && degZ > other.degZ) return true;

return false;

}

bool operator<(const TMonom& other)

{

if (degX < other.degX) return true;

else if (degX == other.degX && degY < other.degY) return true;

else if (degX == other.degX && degY == other.degY && degZ < other.degZ) return true;

return false;

}

bool EqualCoef(const TMonom& other)

{

return degX == other.degX && degY == other.degY && degZ == other.degZ && coef == other.coef;

}

};

## TPolinom.h

#pragma once

#include "THeadList.h"

#include "TMonom.h"

#include <string>

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);

void AddMonom(TMonom newMonom);

TPolinom MultMonom(TMonom monom);

TPolinom operator+(TPolinom& other);

TPolinom AddPolinom(TPolinom& other);

TPolinom operator\*(double coef);

TPolinom operator\* (TPolinom& other);

bool operator==(TPolinom& other);

string ToString();

};

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

{

TMonom m;

m.coef = 1;

m.degX = m.degY = 0, m.degZ = -1;

pHead->value = m;

}

TPolinom::TPolinom(TPolinom& other)

{

TMonom m(1, 0, 0, -1);

pHead->value = m;

for (other.Reset(); !other.IsEnd(); other.GoNext())

{

InsertLast(other.GetCurrentItem());

}

}

inline TPolinom::TPolinom(string str)

{

int i = 0;

while (i < str.length())

{

if (str[i] == ' ' || str[i] == '+') {i++; continue;}

string coef;

for (; isdigit(str[i]) || str[i] == ' ' || str[i] == ',' || str[i] == '-'; i++)

{

if (str[i] == ' ') continue;

coef.push\_back(str[i]);

}

string degX;

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

{

i += 2;

while (isdigit(str[i]))

{

degX.push\_back(str[i]);

i++;

}

}

string degY;

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

{

i += 2;

while (isdigit(str[i]))

{

degY.push\_back(str[i]);

i++;

}

}

string degZ;

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

{

i += 2;

while (isdigit(str[i]))

{

degZ.push\_back(str[i]);

i++;

}

}

double \_coef = 1; int \_degX = 0, \_degY = 0, \_degZ = 0;

if (!coef.empty()) \_coef = stod(coef);

if (!degX.empty()) \_degX = stoi(degX);

if (!degY.empty()) \_degY = stoi(degY);

if (!degZ.empty()) \_degZ = stoi(degZ);

AddMonom(TMonom(\_coef, \_degX, \_degY, \_degZ));

}

}

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

{

while (pFirst != pStop)

{

TNode<TMonom>\* tmp = pFirst;

pFirst = pFirst->pNext;

delete tmp;

}

pFirst = pLast = pPrevious = pCurrent = pStop = pHead;

length = 0;

for (other.Reset(); !other.IsEnd(); other.GoNext())

{

AddMonom(other.GetCurrentItem());

}

return \*this;

}

void TPolinom::AddMonom(TMonom m)

{

Reset();

while (m < pCurrent->value)

{

GoNext();

}

if (pCurrent->value == m)

{

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

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

{

DeleteCurrent();

}

}

else

{

InsertCurrent(m);

}

}

inline TPolinom TPolinom::MultMonom(TMonom monom)

{

TPolinom result;

for (Reset(); !IsEnd(); GoNext())

{

TMonom resultMonom = GetCurrentItem();

resultMonom.coef \*= monom.coef;

resultMonom.degX += monom.degX;

resultMonom.degY += monom.degY;

resultMonom.degZ += monom.degZ;

result.AddMonom(resultMonom);

}

return result;

}

TPolinom TPolinom::operator+(TPolinom& other)

{

TPolinom result(other);

Reset(); result.Reset();

while (!IsEnd())

{

if (result.pCurrent->value > pCurrent->value)

{

result.GoNext();

}

else if (result.pCurrent->value < pCurrent->value)

{

result.InsertCurrent(pCurrent->value);

GoNext();

}

else

{

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

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

{

result.DeleteCurrent();

GoNext();

}

else

{

result.GoNext();

GoNext();

}

}

}

return result;

}

inline TPolinom TPolinom::AddPolinom(TPolinom& other)

{

TPolinom result(other);

for (Reset(); !IsEnd(); GoNext())

result.AddMonom(GetCurrentItem());

return result;

}

inline TPolinom TPolinom::operator\*(double coef)

{

TPolinom result(\*this);

for (Reset(), result.Reset(); !IsEnd(); GoNext(), result.GoNext())

{

TMonom currentMonom = GetCurrentItem();

currentMonom.coef \*= coef;

result.SetCurrentItem(currentMonom);

}

return result;

}

inline TPolinom TPolinom::operator\*(TPolinom& other)

{

TPolinom result;

for (other.Reset(); !other.IsEnd(); other.GoNext())

result = result + MultMonom(other.GetCurrentItem());

return result;

}

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

{

if (GetLength() != other.GetLength()) return false;

for (Reset(), other.Reset(); !IsEnd(); GoNext(), other.GoNext())

if (!GetCurrentItem().EqualCoef(other.GetCurrentItem())) return false;

return true;

}

inline string TPolinom::ToString()

{

string result;

for (Reset(); !IsEnd(); GoNext())

{

TMonom monom = GetCurrentItem();

string coef = to\_string(monom.coef);

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

coef.pop\_back();

result += ' ' + coef;

if (monom.degX != 0) result += "x^" + to\_string(monom.degX);

if (monom.degY != 0) result += "y^" + to\_string(monom.degY);

if (monom.degZ != 0) result += "z^" + to\_string(monom.degZ);

result += ' ';

if (pCurrent->pNext != pStop) result += '+';

}

return result;

}

## TPolinom\_Test.cpp

#include <iostream>

#include "TList.h"

#include "TPolinom.h"

using namespace std;

void TList\_Test()

{

TList<int> list;

cout << "\t===== Тестирование класса TList =====";

cout << "\n1) Заполним список числами от 0 до 9.";

for (int i = 0; i < 10; i++) list.InsertFirst(i);

cout << "\nСписок: ";

for (list.Reset(); !list.IsEnd(); list.GoNext())

cout << list.GetCurrentItem() << " ";

cout << "\n2) Удалим первый элемент, затем удалим элемент со значением 5.";

list.DeleteFirst();

for (list.Reset(); list.GetCurrentItem() != 5; list.GoNext());

list.DeleteCurrent();

cout << "\nСписок: ";

for (list.Reset(); !list.IsEnd(); list.GoNext())

cout << list.GetCurrentItem() << " ";

cout << "\n3) Удалим первый и последний элемент через DeleteCurrent().";

list.Reset();

list.DeleteCurrent();

int k = 0;

for (list.Reset(); k < list.GetLength() - 1; list.GoNext(), k++);

list.DeleteCurrent();

cout << "\nСписок: ";

for (list.Reset(); !list.IsEnd(); list.GoNext())

cout << list.GetCurrentItem() << " ";

cout << "\n4) Присвоим значение 11 третьему элементу и вставим перед ним элемент 10";

list.Reset();

for (int i = 1; i < 3; i++) list.GoNext();

list.SetCurrentItem(11);

list.InsertCurrent(10);

cout << "\nСписок: ";

for (list.Reset(); !list.IsEnd(); list.GoNext())

cout << list.GetCurrentItem() << " ";

}

void TPolinom\_Test()

{

cout << "\t===== Тестирование класса TPolinom =====\n";

TPolinom p1;

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

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

cout << "P1 = " + p1.ToString() + '\n';

TPolinom p2;

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

p2.AddMonom(TMonom(-1, 1, 1, 1));

cout << "P2 = " << p2.ToString() << '\n';

cout << endl;

cout << "P1 + P2 = " << (p1 + p2).ToString();

cout << "\nP2 \* 2 = " << (p2\*2).ToString() << '\n';

cout << "P2 \* 2x^2y^2z^2 = " << (p2.MultMonom(TMonom(2, 2, 2, 2))).ToString() << '\n';

cout << "P1 \* P2 = " << (p1 \* p2).ToString() << '\n';

cout << "\nПроверка ввода полиномов через строку:\n";

TPolinom p3("1,2x^2y^3z^2 + 1,5y^2z^3 + 1,5y^2z^3 + 0,34x^2 + 130,3242z^1");

cout << p3.ToString();

}

int main()

{

setlocale(LC\_ALL, "ru");

TPolinom\_Test();

}

## GoogleTest – TestPolinom.cpp

#include "pch.h"

#include "../Lab-Polynomials/TPolinom.h"

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

TPolinom polinom;

EXPECT\_TRUE(polinom.IsEmpty());

}

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

TPolinom polinom;

TMonom monom(1, 1, 1, 1);

polinom.AddMonom(monom);

EXPECT\_TRUE(!polinom.IsEmpty());

}

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

TPolinom polinom;

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

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

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

polinom.Reset();

EXPECT\_TRUE(polinom.GetLength() == 1 && polinom.GetCurrentItem().coef == 6);

}

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

TPolinom polinom;

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

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

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

polinom.Reset();

EXPECT\_TRUE(polinom.GetLength() == 3);

}

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\_TRUE(polinom.IsEmpty());

}

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

TPolinom polinom;

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

}

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

bool result = true;

TPolinom copiable;

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

copiable.AddMonom(TMonom(2, 1, 1, 2));

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

TPolinom copied(copiable);

EXPECT\_TRUE(copied == copiable);

}

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

bool result = true;

TPolinom assinable;

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

assinable.AddMonom(TMonom(2, 1, 1, 2));

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

TPolinom assigned;

assigned = assinable;

EXPECT\_TRUE(assigned == assinable);

}

TEST(Polinom, Correct\_adding\_operator) {

TPolinom polinom1;

polinom1.AddMonom(TMonom(-1, 1, 1, 1));

polinom1.AddMonom(TMonom(2, 1, 1, 2));

polinom1.AddMonom(TMonom(3, 1, 2, 1));

TPolinom polinom2;

polinom2.AddMonom(TMonom(1, 1, 1, 1));

polinom2.AddMonom(TMonom(2, 1, 1, 2));

polinom2.AddMonom(TMonom(3, 1, 2, 1));

TPolinom result = polinom1 + polinom2;

result.Reset();

float coef1 = result.GetCurrentItem().coef;

result.GoNext();

float coef2 = result.GetCurrentItem().coef;

EXPECT\_TRUE(result.GetLength() == 2 && coef1 == 6 && coef2 == 4);

}

TEST(Polinom, Correct\_constant\_multiply) {

TPolinom p;

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

p.AddMonom(TMonom(2, 1, 1, 2));

TPolinom ans;

ans.AddMonom(TMonom(-2, 1, 1, 1));

ans.AddMonom(TMonom(4, 1, 1, 2));

EXPECT\_TRUE(p\*2 == ans);

}

TEST(Polinom, Correct\_monom\_multiply) {

TPolinom p;

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

p.AddMonom(TMonom(-1, 2, 2, 2));

TPolinom ans;

ans.AddMonom(TMonom(2, 2, 3, 4));

ans.AddMonom(TMonom(-2, 3, 4, 5));

EXPECT\_TRUE(p.MultMonom(TMonom(2,1,2,3)) == ans);

}

TEST(Polinom, Correct\_polinom\_enter) {

TPolinom p = "1x^1y^2z^3 + -2x^3y^2z^1";

TPolinom ans;

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

ans.AddMonom(TMonom(-2, 3, 2, 1));

EXPECT\_TRUE(p == ans);

}

TEST(Polinom, Correct\_multiply\_polinom1) {

bool result = true;

TPolinom p1 = "1x^1y^2z^3 + -2x^3y^2z^1";

TPolinom p2 = "1x^1z^1 + 2y^1";

TPolinom ans;

ans.AddMonom(TMonom(1, 2, 2, 4));

ans.AddMonom(TMonom(-2, 4, 2, 2));

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

ans.AddMonom(TMonom(-4, 3, 3, 1));

EXPECT\_TRUE(p1\*p2 == ans);

}

TEST(Polinom, Correct\_multiply\_polinom2) {

bool result = true;

TPolinom p1 = "3x^2y^3z^2 + 4y^1z^2";

TPolinom p2 = "5z^1 + -1x^3y^2";

TPolinom ans;

ans.AddMonom(TMonom(15, 2, 3, 3));

ans.AddMonom(TMonom(-3, 5, 5, 2));

ans.AddMonom(TMonom(20, 0, 1, 3));

ans.AddMonom(TMonom(-4, 3, 3, 2));

EXPECT\_TRUE(p1 \* p2 == ans);

}

TEST(Polinom, Correct\_multiply\_polinom3) {

bool result = true;

TPolinom p1 = "2x^1-3y^1+4z^1";

TPolinom p2 = "2x^1y^1 - 1y^1z^1 + 1x^1z^2";

TPolinom ans;

ans.AddMonom(TMonom(4, 2, 1, 0));

ans.AddMonom(TMonom(2, 2, 0, 2));

ans.AddMonom(TMonom(-6, 1, 2, 0));

ans.AddMonom(TMonom(-3, 1, 1, 2));

ans.AddMonom(TMonom(6, 1, 1, 1));

ans.AddMonom(TMonom(4, 1, 0, 3));

ans.AddMonom(TMonom(3, 0, 2, 1));

ans.AddMonom(TMonom(-4, 0, 1, 2));

EXPECT\_TRUE(p1 \* p2 == ans);

}