МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное автономное образовательное учреждение

высшего образования

**«Национальный исследовательский**

**Нижегородский государственный университет им. Н.И. Лобачевского»**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения суперкомпьютерных технологий**

Направление подготовки «Прикладная математика и информатика»

**ОТЧЕТ**

по лабораторной работе

**«Арифметические операции с полиномами»**

**Выполнил:** студент группы 381603-1  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Хорькин А.С

**Проверила:** к.т. н., старший преподаватель каф. МОСТ института ИТММ  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кустикова В.Д.

Нижний Новгород

2018

**Содержание**

[Введение 3](#_Toc515580395)

[1 Постановка задачи 4](#_Toc515580396)

[2 Руководство пользователя 5](#_Toc515580397)

[3 Руководство программиста 6](#_Toc515580398)

[3.1 Описание структуры программы 6](#_Toc515580399)

[3.2 Описание структуры данных 7](#_Toc515580400)

[3.3 Описание программной реализация 10](#_Toc515580401)

[Заключение 13](#_Toc515580402)

[Список литературы 14](#_Toc515580403)

[Приложение 15](#_Toc515580404)

[Приложение A. Node.h 15](#_Toc515580405)

[Приложение B. List.h 15](#_Toc515580406)

[Приложение C. Monom.h 19](#_Toc515580407)

[Приложение D. Polinom.h 19](#_Toc515580408)

[Приложение E. Polinom.cpp 20](#_Toc515580409)

[Приложение F. main.cpp 26](#_Toc515580410)

[Приложение G. test\_polinom.cpp. Тесты программы 28](#_Toc515580411)

# Введение

В настоящий момент многочлены играют большую роль в математике: в частности в теории кодирования, алгебраической геометрии, топологии, теории узлов и т.д. Вследствие этого разработка программы, выполняющую различные операцици с многочленами является важной и востребованной задачей. Необходимо реализовать программу, которая умеет делать некоторые арифметические операции над полиномами, имеющими три переменные и степень от 0 до 9.Данная программа может помочь как при обыкновенном подсчете нужных пользователю полиномов, так и при разработке других программ в которой данная программа является небольшой составной частью. Реализованы полиномы с помощью такой структуры данных как *циклический список* *с головой*. При реализации различных алгоритмов данной программы приходит хорошее понимание как эта структура данных устроена.

# Постановка задачи

Формулировка задачи:

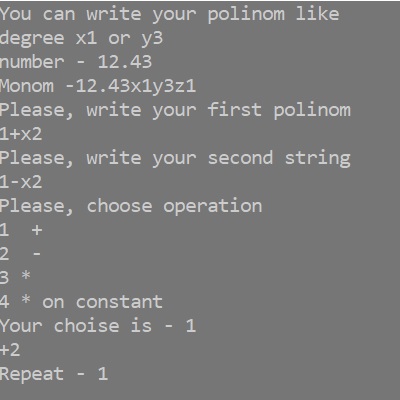
Разработать программу, выполняющую арифметические операции над полиномами от трех переменных. Полином должен быть составлен из мономов трех переменных, а именно: x, y, z со степенями от 0 до 9. Коэффициенты перед мономами – вещественные числа. Допустимые операции: умножение двух полиномов, сумма, вычитание, умножение полинома на константу.

Исходные данные:

2 строки, содержащие полином A и полином B соответственно.

Требуемый результат:

Программа правильно производит арифметические операции, которые нужны пользователю

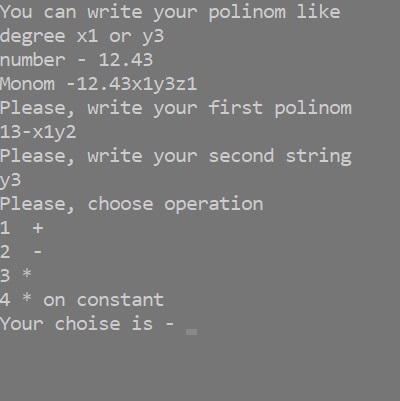


1. Пример работы программы.

# Руководство пользователя

Для начала работы программы необходимо открыть файл Sample.exe

После чего программа попросит пользователя ввести первый полином, а затем второй. Какие символы использовать написано вначале.

Далее пользователь вправе выбрать одну из предложенных операций. 

1. Выбор операции

После пользователю выводится результат. В конце работы программы вам будет предложено либо продолжить вычисления, либо завершить ее. Для продолжения необходимо нажать 1, для завершения – любую другую клавишу.

# Руководство программиста

## Описание структуры программы

Структура решения:

* gtest (статическая библиотека Гугл тестов)
* Polinom (статическая библиотека для полиномов)
* Sample (файл main.cpp)
* Test (консольное приложение, запускающее тесты)

**Библиотека gtest** состоит из:

gtest.h заголовочный файл для Гугл тестов

gtest – all.cpp source файл для Гугл тестов

**Библиотека Polinom** состоит из:

monom.h – заголовочный файл, который содержит специальную структуру данных, предназначенную для хранения слагаемый многочлена.

node.h – шаблонный класс, стандартная структура данных, необходимая для листа

list.h – шаблонный класс, циклический список с головой

polinom.h – содержит list<monom> List

Файлы .cpp содержат реализацию, за исключением шаблонных классов.

**Проект Sample** состоит из:

main\_polinom.cpp – файл, содержащий main() функцию

**Проект test** состоит из:

Test\_main.cpp – содержит функцию main(), запускающую Гугл тесты.

Test\_polinom.cpp – содержит разнообразные тесты для всех структур данных, реализованных в программе.

## Описание структуры данных

В программе используется несколько структур данных или классов: класс мономов, класс ноды, класс кольцевой лист с головой, класс полиномов. Моном состоит из двух полей, а именно double f – коэффициент каждого звена и int abc – хранит в себе степень звена.Таким образом, что степень x это сотый разряд, степень y десятый разряд, а степень z – первый разряд. Устанавливается биекция между числами от 0 до 999 и всевозможными степенями монома от трех переменных. Нода – шаблонный класс, стандартно состоит из двух полей: полей данных и указателя на следующий объект этого же класса, то есть на следующую ноду. Класс кольцевой лист состоит из указателя на первую ноду называемую головой и указателя на текущую ноду. Последняя нода указывает на следующую ноду после головы. Полином же состоит из одного поля – Лист, ноды которого хранят структуру данных моном.

Описание алгоритмов, применяющихся в программе.

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

Парсинг строки в полином

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

Ввод полинома осуществляется с клавиатуры, необходимо преобразовать строку полинома в структуру данных - Polinom, такой процесс называется парсингом.

Для такого преобразования используется

1) Алгоритм преобразования исходной строки в список, ноды которого хранят строку, которая является одним мономом. Порядок следования мономов такой же как и в исходной строке.

2)Список из строк преобразуется в список из мономов в нужном порядке (Убывание степеней).

Таким образом алгоритм перевода строки в полином можно назвать двухуровневым парсингом. На первом этапе из строки создается список строк мономов. На втором этапе из списка строк мономов создается список из мономов в необходимом порядке что и составляет полином. Такая реализация обеспечивает разделение логики выделения мономов и их упорядочивание.

Парсинг первого уровня

1) На вход поступает строка, введенная пользователем. Строка проходится слева-направо до тех пор, пока не встретится символ '+', '-' или не случится конец строки.

Как только мы встретили символ, мы выделяем подстроку до этого символа кладём её в список.

Парсинг второго уровня.

2) На вход поступает список из строк - мономов.

Сначала в списке ищется моном, чья степень максимальна. Степень монома представляет собой число от 0 до 999. Чтобы выяснить степень монома (пока что это строка), её надо перевести из структуры строка в структуру моном. Тут происходит поиск коэффициента и степеней x, y, z. Найденные значения записываются в структуру Monom, таким образом, что степень x это сотый разряд, степень y десятый разряд, а степень z – первый разряд. Таким образом int abc хранит степень монома. В f – записывается коэффициент.

После нахождения максимального монома в списке из строк. Мы переводим эту строку в структуру данных моном и записываем её в конец результирующего списка из мономов, составляющего сущность полинома. Из списка строк нода с этим мономом удаляется. Процесс повторяется до тех пор, пока в списке из строк не останется ни одной ноды содержащей данные. Попросту говоря пока список из строк не станет пустым.

Пустой (нулевой) полином это список из нулевого монома, который указывает сам на себя.

таким образом формируется упорядоченный список из мономов.

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

Сложение полиномов

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

Сложение полиномов осуществляется алгоритмом слияния упорядоченных массивов. В данном случае он применяется к упорядоченным спискам:

На вход подаются два упорядоченных списка.

Сравниваем элементы списков:

Больший(Моном у которого большая степень)) элемент вставляем в результат. Двигаем указатель списка, из которого взяли элемент, на одно звено вперед.

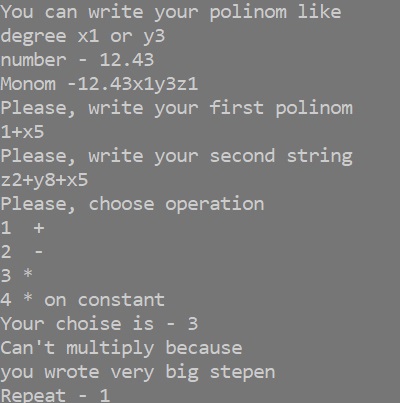
В случае равенства, складываем коэффициенты и записываем в результат. Двигаем указатели обоих списков на одно звено вперед.

После того, как один из списков кончился, записываем остаток второго списка в результат.

Вычитание полиномов выполняется через сложение с умножением на константу:

C = A - B = A + (-1) \*B

Если при умножении полиномов полученные степени переменных больше 9, выводится сообщение об ошибке.



1. Сообщение об ошибке

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

Умножение полиномов

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

Сначала берется первый моном первого полинома и умножается на другой полином, в результате получается полином. Затем его нужно сложить с результатом произведения следующего монома первого полинома умноженного на второй полином и т.д. Алгоритм заканчивается, когда будут использованы все мономы первого полинома.

## Описание программной реализация

Node:

**node – шаблонная структура, является узлом списка, с параметром Type1 – тип данных, хранящийся в узле.**

***Поля:***

Type1 date – данные в узле

node\* next– указатель на следующий узел

***Методы:***

node() { next = NULL; } – конструктор по умолчанию

node(const Type1 date, node\* next = NULL) - конструктор с параметрами

node(const node<Type1> &con) - консткруктор копирования

node(const node\* con) - конструктор копирования по указателю

Лист:

**list – шаблонный класс, циклический список с “головой”, с параметром T – тип данных, хранящийся в списке**

***Поля:***

node<T>\* current; – указатель на текущий

node<T>\* Head; – указатель на голову

Методы:

list(); - конструктор

list(const list<T> &list2); - конструктор с параметрами

~list(); -деструктор

list<T>& operator = (const list<T> &list2); – оператор присваивания

bool IsEmpty() const; – проверка на пустоту стиска

void Insert\_to\_tail(const T con = 0); – вставка в конец

void clear();– очистка списка

bool operator == (const list<T> &list2) const; – оператор сравнения

bool operator != (const list<T> &list2) const; – оператор сравнения

void reset(); - установка на начало списка

void Dellcurrent(); - удаление текущей ноды

void getnext();– сдвиг указателя на следующий узел

T getcurrentdate(); - получить значение текущих данных

T getcurrentnextdata(); - получить значение следующих данных

bool IsEnd();проверка на окончания списка

Monom:

**monom –класс, представляет собой моном, из которых составлен полином**

***Поля:***

int abc; – степень монома

double f; – коэффициент монома

Методы:

Monom(const string &A);

Monom(double ff = 0, int abcc = 0) - конструктор с параметрами

void operator = (const Monom &A) - конструктор копирования

bool operator ==(const Monom &A) const - оператор сравнения

bool operator !=(const Monom &A) const - оператор сравнения

bool operator < (const Monom &A) const - оператор сравнения

bool operator > (const Monom &A) const - оператор сравнения

Monom operator \* (const Monom&A) const - оператор умножения монома на моном

Класс полином:

**Polynom –класс, представляет собой полином**

***Поля:***

list<Monom> mnogochlen;

***Методы:***

polinom() - конструктор по умолчанию

polinom(const string &a) - конструктор с параметрами

polinom(const polinom &copy) - конструктор копирования

polinom operator + (const polinom &polinom2) - оператор сложения

polinom operator - (const polinom &polinom2) - оператор вычитания

polinom& operator = (const polinom &polinom2) - оператор присваивания

polinom operator \* (const polinom &polinom2) - оператр умножения на полином

polinom operator \* (const Monom Mon) - оператор умножения на моном

bool operator == (const polinom &polinom2) const - оператор сравнения

bool operator != (const polinom &polinom2) const - оператор сравнения

bool can\_multyply(Monom A, Monom B) - возможность умножения мономов

private методы:

list<string>\* parsing\_to\_string(const string &a) - парсинг первого уровня

void Create\_polinom(list<string>\* a) - парсинг второго уровня

Monom parsing\_to\_monom(string ptr) - преобразование строки в моном

# Заключение

В лабораторной работе была реализована программа, выполняющая арифметические операции с полиномами. Был разработан класс полиномов, на основе циклического односвязного списка с головой. Таким образом программа может выполнять все необходимые операции с полиномами.

# Список литературы

1. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К*. Алгоритмы: построение и анализ, Москва 2013.
2. Рабочие материалы к учебному курсу «Методы программирования». *Гергель В.П*. 2002 г.

# Приложение

## Приложение A. Node.h

#pragma once

#include <string>

template <typename Type1>

class node {

public:

Type1 date;

node\* next;

node() { next = NULL; };

node(const Type1 date, node\* next = NULL);

node(const node<Type1> &con);

node(const node\* con);

};

template <typename Type1>

node<Type1>::node(const Type1 date, node\* next) {

this->next = next;

this->date = date;

}

template <typename Type1>

node<Type1>::node(const node<Type1> &con) {

date = con.date;

}

template <typename Type1>

node<Type1>::node(const node\* con) {

date = con->date;

}

## Приложение B. List.h

#pragma once

#define NULL 0

#include "node.h"

/\*СПИСОК С ГОЛОВОЙ\*/

template <typename T>

class list {

private:

node<T>\* current;

node<T>\* Head;

public:

list();

list(const list<T> &list2);

~list();

list<T>& operator = (const list<T> &list2);

bool IsEmpty() const;

void Insert\_to\_tail(const T con = 0);

void clear();

bool operator == (const list<T> &list2) const;

bool operator != (const list<T> &list2) const;

void Dellcurrent();

void reset();

void getnext();

T getcurrentdate();

T getnextcurrentdate();

bool IsEnd();

void Delete(const T date);

};

template <typename T>

T list<T>:: getnextcurrentdate() {

return current->next->date;

}

template <typename T>

void list<T>:: Delete(const T date1) {

if (!IsEmpty()) {

int a = 1;

node<T> \* ptr = Head->next;

if (ptr->date != date1) {

ptr = ptr->next;

}

else a = 0;

while ((ptr->date != date1)&&(ptr!=Head->next)&&(a==1)) ptr = ptr->next;

if (a == 1 && ptr == Head->next) throw "netu node with this date";

//нашли адрес ноды которую нужно удалить

if (a == 0) {

//это первая нода надо переобозначить Head->next; и

node<T> \* ptrend = Head->next;

int flag = 0;

while (ptrend->next != Head->next) {

ptrend = ptrend->next;

flag = 1;

}

if (flag == 1) {

node<T> \*p = ptr->next;

delete ptr;

Head->next = p;

current = Head->next;

ptrend->next = Head->next;

}

else {

delete ptr;

Head->next = Head;

current = NULL;

}

}

else

{

node<T> \*pprev = Head->next;

node<T> \*pnext = ptr->next;

while (pprev->next != ptr) pprev = pprev->next;

delete ptr;

pprev->next = pnext;

current = pprev;

}

}

else throw"cant delete empty list";

}

template <typename T>

bool list<T>::IsEnd() {

if (current->next == Head->next) return true; else return false;

}

template <typename T>

void list<T>:: getnext() {

current = current->next;

}

template <typename T>

T list<T>:: getcurrentdate() {

return current->date;

}

template <typename T>

void list<T>:: reset() {

if (!IsEmpty()) current = Head->next;

}

template <typename T>

list<T>::list() {

Head = new node<T>;

Head->next = Head;

current = NULL;

}

template <typename T>

bool list<T>::IsEmpty() const {

if (current == NULL)

return true;

else

return false;

}

template <typename T>

void list<T>::Insert\_to\_tail(const T con) {

if (!IsEmpty()) {

while (current->next != Head->next) current = current->next;

current->next = new node<T>(con, Head->next);

}

else {

Head->next = new node<T>(con);

Head->next->next = Head->next;

current = Head->next;

}

}

template <typename T>

list<T>::list(const list<T> &list2) {

Head = new node<T>;

Head->next = Head;

current = NULL;

if (!list2.IsEmpty()) {

node<T>\* ptr2 = list2.Head;//указывает на фиктивную list2 ноду

ptr2 = ptr2->next;// нода с данными у того от куда копирую

Head->next = new node<T>(ptr2);

node<T>\* ptrend = ptr2;

current = Head->next; // current на ноду с данными у исходного

while (ptrend->next!= ptr2) ptrend = ptrend->next;

ptr2 = ptr2->next;

while (ptr2 != list2.Head->next) {

current->next = new node<T>(ptr2);

current = current->next;

ptr2 = ptr2->next;

}

//зациклить

current->next = Head->next;

}

current = list2.current;

}

template <typename T>

list<T>::~list() {

this->clear();

}

template <typename T>

void list<T>::clear(){

if (!IsEmpty()) {

reset();

node<T>\* end;

while (!IsEnd()) getnext();

end = current;

reset();

while (!IsEnd()) {

node<T>\* ptr = current->next;

delete current;

Head->next = ptr;

current = ptr;

end->next = ptr;

}

delete current;

current = NULL;

Head->next = Head;

}

}

template <typename T>

list<T>& list<T>:: operator = (const list<T> &list2) { //необходимо обходиться без итераторов

this->clear();

if (!list2.IsEmpty()) {

//найдём конец у list2

node<T>\* ptr2 = list2.Head;//указывает на фиктивную list2 ноду

ptr2 = ptr2->next;// нода с данными у того от куда копирую

Head->next = new node<T>(ptr2);

node<T>\* ptrend = ptr2;

current = Head->next; // current на ноду с данными у исходного

while (ptrend->next != ptr2) ptrend = ptrend->next;

ptr2 = ptr2->next;

while (ptr2 != list2.Head->next) {

current->next = new node<T>(ptr2);

current = current->next;

ptr2 = ptr2->next;

}

//зациклить

current->next = Head->next;

}

return (\*this);

}

template <typename T>

bool list<T> ::operator == (const list<T> &list2) const {

bool res = true;

if (list2.IsEmpty() && this->IsEmpty()) return true;

else {

node<T>\* listc = Head->next; //Адрес первой носящей смысл ноды с даннными

node<T>\* list2c = list2.Head->next; //Адрес первой носящей смысл ноды с даннными

if (listc->date != list2c->date) res = false;

listc = listc->next;

list2c = list2c->next;

while (listc != Head->next && list2c != list2.Head->next) {

if (listc->date != list2c->date) res = false;

listc = listc->next;

list2c = list2c->next;

}

if (listc != Head->next || list2c != list2.Head->next) res = false;

}

return res;

}

template <typename T>

bool list<T> ::operator != (const list<T> &list2) const {

if (\*this == list2) return 0; else return 1;

}

template <typename T>

void list<T> ::Dellcurrent()

{

if (!this->IsEmpty())

{

node<T> \* Ptr = Head->next;

node<T> \* Ptrnext = current->next;

node<T> \* Ptr2 = Head->next;

if ((current == Ptr) && (current->next == Ptr)) //надо сделать список пустым

{

delete current;

Head->next = Head;

current = NULL;

}

else if ((current == Ptr) && (current->next != Ptr)) //удалить начальный

{

while (Ptr2->next != Head->next) Ptr2 = Ptr2->next; // теперь Ptr2 - конец

delete current;

Head->next = Ptrnext;

current = Ptrnext;

Ptr2->next = Head->next;

}

else if (current != Ptr) // удалить не начальный, но конечный или средний

{

while (Ptr->next != current) Ptr = Ptr->next;

Ptr->next = Ptrnext;

delete current;

current = Ptr;

}

}

}

## Приложение C. Monom.h

#pragma once

#include <string>

using namespace std;

class Monom {

public:

int abc;

double f;

Monom(const string &A);

Monom(double ff = 0, int abcc = 0);

void operator = (const Monom &A);

bool operator ==(const Monom &A) const;

bool operator !=(const Monom &A) const;

bool operator < (const Monom &A) const;

bool operator > (const Monom &A) const;

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

};

## Приложение D. Polinom.h

#include "monom.h"

#include "list.h"

#include <string>

using namespace std;

class polinom

{

private:

list<Monom> mnogochlen;

list<string>\* parsing\_to\_string(const string &a); //parsing first level

void Create\_polinom(list<string>\* a); //parsing second level

Monom parsing\_to\_monom(string ptr);

public:

polinom();

polinom(const string &a);

polinom(const polinom &copy);

polinom operator + (const polinom &polinom2);

polinom operator - (const polinom &polinom2);

polinom& operator = (const polinom &polinom2);

polinom operator \* (const polinom &polinom2);

polinom operator \* (const Monom Mon);

bool operator == (const polinom &polinom2) const;

bool operator != (const polinom &polinom2) const;

bool can\_multyply(Monom A, Monom B);

friend ostream& operator<<(ostream& os, const polinom& p);

friend polinom operator \* (const polinom& A, const double a);

friend polinom operator \* (const double a, const polinom& A);

};

## Приложение E. Polinom.cpp

#include "polinom.h"

#include<iostream>

#include "list.h"

#include<monom.h>

#include<ostream>

#include<string>

using namespace std;

polinom::polinom() :mnogochlen() {

mnogochlen.Insert\_to\_tail();

}

polinom::polinom(const string &a): mnogochlen() {

list<string> \* liststr = parsing\_to\_string(a);//спосок из листов

Create\_polinom(liststr); //создание полинома по этому списку

}

list<string>\* polinom:: parsing\_to\_string(const string &A) {

list<string>\* ListMonomStr = new list<string>;

int i=0;

string monomstr="";

while (i < A.length()) {

switch (A[i])

{

case '-': {

if (i == 0) { monomstr.push\_back(A[i++]); }

else {

ListMonomStr->Insert\_to\_tail(monomstr);

monomstr = "";

monomstr.push\_back('-');

i++;

}

break;

}

case '+': {

ListMonomStr->Insert\_to\_tail(monomstr);

monomstr = "";

i++;

break;

}

default: {

monomstr.push\_back(A[i++]);

}

break;

}

}

ListMonomStr->Insert\_to\_tail(monomstr);

return ListMonomStr;

}

Monom polinom:: parsing\_to\_monom(string ptr) {

Monom D(ptr);

return D;

}

void polinom::Create\_polinom(list<string>\* a) {

a->reset();

while (!a->IsEmpty()) {

//находим максимальную ноду и вставляем её в конец

string res;

a->reset();

res = a->getcurrentdate();

a->getnext();

while (!a->IsEnd()) {

if (parsing\_to\_monom(res) < parsing\_to\_monom(a->getcurrentdate())) res = a->getcurrentdate();

a->getnext();

}

if (parsing\_to\_monom(res) < parsing\_to\_monom(a->getcurrentdate())) res = a->getcurrentdate();

a->getnext();

//нашли максимальную ноду и вставляем её в конец

mnogochlen.Insert\_to\_tail(parsing\_to\_monom(res));

a->Delete(res);

}

}

Monom parsing\_to\_monom(string ptr) {

Monom D(ptr);

return D;

}

polinom::polinom(const polinom &copy) :mnogochlen() {

mnogochlen = copy.mnogochlen;

}

polinom& polinom:: operator = (const polinom &polinom2) {

mnogochlen = polinom2.mnogochlen;

return(\*this);

}

polinom polinom:: operator + (const polinom &polinom2) {

polinom res;

Monom w;

res.mnogochlen.Delete(w);

polinom copy(polinom2);

copy.mnogochlen.reset();

mnogochlen.reset();

//начало алгоритма //проверяем следующие

bool exit = false;

bool forone = true;

int flag = 0; // флаг на у кого в конце

int flag2 = 1; //флаг на единственный моном в полиноме copy

int flag3 = 1; //флаг на единственный моном в полиноме this

if(copy.mnogochlen.IsEnd()) flag2 = 0;

if(mnogochlen.IsEnd()) flag3 = 0;

while ((!copy.mnogochlen.IsEnd())&&(!mnogochlen.IsEnd())) {

if (copy.mnogochlen.getcurrentdate() < mnogochlen.getcurrentdate()) {

res.mnogochlen.Insert\_to\_tail(mnogochlen.getcurrentdate());

mnogochlen.getnext();

}

else if (copy.mnogochlen.getcurrentdate() > mnogochlen.getcurrentdate()) {

res.mnogochlen.Insert\_to\_tail(copy.mnogochlen.getcurrentdate());

copy.mnogochlen.getnext();

}

else {

if (copy.mnogochlen.getcurrentdate().f + mnogochlen.getcurrentdate().f != 0) {

res.mnogochlen.Insert\_to\_tail(Monom(copy.mnogochlen.getcurrentdate().f + mnogochlen.getcurrentdate().f, mnogochlen.getcurrentdate().abc));

}

mnogochlen.getnext();

copy.mnogochlen.getnext();

}

}

//флаг равен 1 если конец у copy

if ((copy.mnogochlen.IsEnd()) && (!mnogochlen.IsEnd())) flag = 1;

if ((copy.mnogochlen.IsEnd()) && (mnogochlen.IsEnd())) exit = true;

while (exit==false) { //когда у одного в конце надо проверять все с ним

if (copy.mnogochlen.getcurrentdate() < mnogochlen.getcurrentdate()) {

if (flag == 0) exit = true;

res.mnogochlen.Insert\_to\_tail(mnogochlen.getcurrentdate());

mnogochlen.getnext();

if (mnogochlen.IsEnd()) exit = true;

}

else if (copy.mnogochlen.getcurrentdate() > mnogochlen.getcurrentdate()) {

if (flag == 1) exit = true;

res.mnogochlen.Insert\_to\_tail(copy.mnogochlen.getcurrentdate());

copy.mnogochlen.getnext();

if (copy.mnogochlen.IsEnd()) exit = true;

}

else {

if (copy.mnogochlen.getcurrentdate().f + mnogochlen.getcurrentdate().f != 0) {

res.mnogochlen.Insert\_to\_tail(Monom(copy.mnogochlen.getcurrentdate().f + mnogochlen.getcurrentdate().f, mnogochlen.getcurrentdate().abc));

}

exit = true;

if (flag2 == 0 || flag3 == 0) forone = false; //значит последний использовался только что мы его обработали

mnogochlen.getnext();

copy.mnogochlen.getnext();

}

}

//когда они оба в цонце

if ((copy.mnogochlen.IsEnd() && mnogochlen.IsEnd() && forone == true)) {

if (copy.mnogochlen.getcurrentdate() < mnogochlen.getcurrentdate()) {

res.mnogochlen.Insert\_to\_tail(mnogochlen.getcurrentdate());

res.mnogochlen.Insert\_to\_tail(copy.mnogochlen.getcurrentdate());

mnogochlen.getnext();

}

else if (copy.mnogochlen.getcurrentdate() > mnogochlen.getcurrentdate()) {

res.mnogochlen.Insert\_to\_tail(copy.mnogochlen.getcurrentdate());

res.mnogochlen.Insert\_to\_tail(mnogochlen.getcurrentdate());

copy.mnogochlen.getnext();

}

else {

if (copy.mnogochlen.getcurrentdate().f + mnogochlen.getcurrentdate().f != 0) {

res.mnogochlen.Insert\_to\_tail(Monom(copy.mnogochlen.getcurrentdate().f + mnogochlen.getcurrentdate().f, mnogochlen.getcurrentdate().abc));

}

mnogochlen.getnext();

copy.mnogochlen.getnext();

}

}

else if (((flag == 1) && (!copy.mnogochlen.IsEnd()))||(flag2==0)) { //копируем всё у this а

while (!mnogochlen.IsEnd()) {

res.mnogochlen.Insert\_to\_tail(mnogochlen.getcurrentdate());

mnogochlen.getnext();

}

res.mnogochlen.Insert\_to\_tail(mnogochlen.getcurrentdate());

mnogochlen.getnext();

}

else {//копируем всё у copy а

while (!copy.mnogochlen.IsEnd()) {

res.mnogochlen.Insert\_to\_tail(copy.mnogochlen.getcurrentdate());

copy.mnogochlen.getnext();

}

res.mnogochlen.Insert\_to\_tail(copy.mnogochlen.getcurrentdate());

copy.mnogochlen.getnext();

}

if (res.mnogochlen.IsEmpty()) res.mnogochlen.Insert\_to\_tail();

return res;

}

polinom polinom:: operator - (const polinom &polinom2) {

polinom w("-1");

return (\*this)+w\*polinom2;

}

bool polinom:: can\_multyply(Monom A, Monom B) {

bool can = true;

if ((A.abc/100 + B.abc/100) >= 10) can = false;

if(((A.abc/10)%10 + (B.abc/10))%10 >= 10) can = false;

if((A.abc % 10 + B.abc % 10) >= 10) can = false;

return can;

}

polinom polinom:: operator \* (const polinom &polinom2) {

Monom w;

polinom copy(polinom2); //от второго

polinom res;

res.mnogochlen.Delete(w);

copy.mnogochlen.reset();

mnogochlen.reset();

res = (\*this)\*copy.mnogochlen.getcurrentdate();

//проверка что this единственный

while (!copy.mnogochlen.IsEnd()) {

res = res + (\*this)\*copy.mnogochlen.getnextcurrentdate();

copy.mnogochlen.getnext();

}

return res;

}

polinom polinom:: operator \* (const Monom Mon) {

polinom res;

polinom copy(\*this);

res.mnogochlen.Delete(Monom(0, 0));

res.mnogochlen.reset();

copy.mnogochlen.reset();

//первую умножим отдельно

if ((Mon.f == 0) || (copy ==polinom())) {

res.mnogochlen.Insert\_to\_tail();

}

else {

if (can\_multyply(copy.mnogochlen.getcurrentdate(), Mon))

res.mnogochlen.Insert\_to\_tail(copy.mnogochlen.getcurrentdate()\*Mon); else throw "Stepenm vey big";

//перемножили

while (!copy.mnogochlen.IsEnd()) {

if (can\_multyply(copy.mnogochlen.getcurrentdate(), Mon)) {

res.mnogochlen.Insert\_to\_tail(copy.mnogochlen.getnextcurrentdate()\*Mon);

copy.mnogochlen.getnext();

} else throw "Stepenm vey big";

}

}

return res;

}

bool polinom:: operator == (const polinom &polinom2) const {

bool res = true;

polinom A(\*this);

polinom B(polinom2);

A.mnogochlen.reset();

B.mnogochlen.reset();

// первые мономы проверим отдельно

if (A.mnogochlen.getcurrentdate() != B.mnogochlen.getcurrentdate()) res = false;

while (!A.mnogochlen.IsEnd() && !B.mnogochlen.IsEnd()) {

if (A.mnogochlen.getnextcurrentdate() != B.mnogochlen.getnextcurrentdate()) res = false;

A.mnogochlen.getnext();

B.mnogochlen.getnext();

}

if (!A.mnogochlen.IsEnd() || !B.mnogochlen.IsEnd()) res = false;

return res;

}

bool polinom:: operator != (const polinom &polinom2) const {

if (\*this == polinom2) return 0; else return 1;

}

ostream& operator<<(ostream& os, const polinom& P)

{

int flag = 0;

polinom P1(P);

P1.mnogochlen.reset();

//os << "0";

while (!P1.mnogochlen.IsEnd())

{

if (P1.mnogochlen.getcurrentdate().f == 0)

P1.mnogochlen.getnext();

else {

flag = 1;

if (P1.mnogochlen.getcurrentdate().f > 0)

os << '+';

os << P1.mnogochlen.getcurrentdate().f;

string ABC = to\_string(P1.mnogochlen.getcurrentdate().abc);

if (ABC.length() == 1) // если abc =1

{

if (ABC == "0")

P1.mnogochlen.getnext();

else {

os << "z^" << ABC[0];

P1.mnogochlen.getnext();

}

}

else if (ABC.length() == 2) // если abc =2

{

os << "y^" << ABC[0];

if (ABC[1] != '0')

os << "z^" << ABC[1];

P1.mnogochlen.getnext();

}

else if (ABC.length() == 3) // если abc =3

{

os << "x^" << ABC[0];

if (ABC[1] != '0')

os << "y^" << ABC[1];

if (ABC[2] != '0')

os << "z^" << ABC[2];

P1.mnogochlen.getnext();

}

}

}

// последняя итерация цикла

if (P1.mnogochlen.getcurrentdate().f == 0)

P1.mnogochlen.getnext();

else {

flag = 1;

if (P1.mnogochlen.getcurrentdate().f > 0)

os << '+';

os << P1.mnogochlen.getcurrentdate().f;

string ABC = to\_string(P1.mnogochlen.getcurrentdate().abc);

if (ABC.length() == 1) // если abc =1

{

if (ABC == "0")

P1.mnogochlen.getnext();

else {

os << "z^" << ABC[0];

P1.mnogochlen.getnext();

}

}

else if (ABC.length() == 2) // если abc =2

{

os << "y^" << ABC[0];

if (ABC[1] != '0')

os << "z^" << ABC[1];

P1.mnogochlen.getnext();

}

else if (ABC.length() == 3) // если abc =3

{

os << "x^" << ABC[0];

if (ABC[1] != '0')

os << "y^" << ABC[1];

if (ABC[2] != '0')

os << "z^" << ABC[2];

P1.mnogochlen.getnext();

}

}

if (flag == 0)

os << "0";

os << endl;

return os;

}

polinom operator \* (const polinom& A, const double a)

{

polinom Res;

polinom A1(A);

Res.mnogochlen.Delete(Monom(0, 0));

A1.mnogochlen.reset();

while (!A1.mnogochlen.IsEnd())

{

Monom res(A1.mnogochlen.getcurrentdate().f\*a, A1.mnogochlen.getcurrentdate().abc);

if (res.f != 0) Res.mnogochlen.Insert\_to\_tail(res);

A1.mnogochlen.getnext();

}

// последняя итерация цикла

Monom res(A1.mnogochlen.getcurrentdate().f\*a, A1.mnogochlen.getcurrentdate().abc);

if (res.f != 0) Res.mnogochlen.Insert\_to\_tail(res);

A1.mnogochlen.getnext();

if (Res.mnogochlen.IsEmpty()) Res.mnogochlen.Insert\_to\_tail();

return Res;

}

polinom operator \* (const double a, const polinom& A)

{

return A \* a;

}

## Приложение F. main.cpp

#include "monom.h"

#include "polinom.h"

#include <iostream>

#include <ostream>

using namespace std;

void main()

{

int flag = 1;

int k = 0;

while (flag == 1)

{

system("cls");

cout << "You can write your polinom like" << endl;

cout << "degree x1 or y3" << endl<< "number - 12.43"<< endl<< "Monom -12.43x1y3z1"<<endl;

cout << "Please, write your first polinom" << endl;

string s1;

cin >> s1;

polinom S1(s1);

cout << "Please, write your second string" << endl;

string s2;

cin >> s2;

polinom S2(s2);

cout << "Please, choose operation" << endl;

cout << "1 +" << endl;

cout << "2 -" << endl;

cout << "3 \*" << endl;

cout << "4 \* on constant" << endl;

cout << "Your choise is - ";

cin >> k;

switch (k)

{

case 1:

{

polinom res = S1 + S2;

cout << res;

break;

}

case 2:

{

polinom res = S1 - S2;

cout << res;

break;

}

case 3:

{

try {

polinom res = S1 \* S2;

cout << res;

}

catch (...) {

cout << "Can't multiply because"<< endl<< "you wrote very big stepen"<<endl;

}

break;

}

case 4:

{

//string con;

//cin >> con;

double a;

cout << "Please, write const for multyplication" << endl;

cin >> a;

//Polinom C(con);

polinom res = a \* S1;

cout << res;

break;

}

default:

break;

}

cout << "Repeat - 1" << endl;

cin >> flag;

}

}

## Приложение G. test\_polinom.cpp. Тесты программы

#include <gtest.h>

#include "polinom.h"

#include "monom.h"

TEST(list, insert\_to\_end) // вставка в конец

{

Monom a(12, 14);

Monom b(14, 43);

list<Monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.reset();

Monom res(l.getcurrentdate());

EXPECT\_EQ(res, a);

}

TEST(list, can\_create\_list) // может создать лист

{

EXPECT\_NO\_THROW(list<Monom> l);

}

TEST(list, delete\_last\_not\_empty\_list) // Удаление не пустого листа

{

Monom a(12, 14);

Monom b(14, 43);

list<Monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.clear();

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

}

TEST(list, can\_check\_is\_empty) // проверка на пустоту не пустого листа

{

Monom a(12, 14);

Monom b(14, 43);

list<Monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

EXPECT\_EQ(0, l.IsEmpty());

}

TEST(list, can\_check\_is\_empty\_if\_empty) // проверка на пустоту пустого листа

{

list<Monom> l;

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

}

TEST(list, operator\_ravno\_for\_list) // проверка оператора =

{

Monom a(12, 14);

Monom b(14, 43);

list<Monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

list<Monom> z = l;

EXPECT\_EQ(z, l);

}

TEST(list, operator\_ravno\_for\_list2)

{

Monom a(12, 14);

Monom b(14, 43);

list<Monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

list<Monom> z = l;

if (l == z)

EXPECT\_EQ(l, z);

}

TEST(list, operator\_ravno\_for\_list3)

{

Monom a(12, 14);

Monom b(14, 43);

Monom c(11, 123);

Monom d(12, 1232);

list<Monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.Insert\_to\_tail(c);

l.Insert\_to\_tail(d);

list<Monom> z = l;

if (l == z)

EXPECT\_EQ(l, z);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//dellcurent

TEST(list, Dellcurent\_first)

{

Monom a(12, 14);

Monom b(14, 43);

Monom c(11, 123);

Monom d(12, 1232);

list<Monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.Insert\_to\_tail(c);

l.Insert\_to\_tail(d);

l.reset();

l.Dellcurrent();

EXPECT\_EQ(l.getcurrentdate(), b);

}

TEST(list, Dellcurent\_last)

{

Monom a(12, 14);

Monom b(14, 43);

Monom c(11, 123);

Monom d(12, 1232);

list<Monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.Insert\_to\_tail(c);

l.Insert\_to\_tail(d);

l.getnext();

l.Dellcurrent();

EXPECT\_EQ(l.getcurrentdate(), c);

}

TEST(list, Dellcurent\_middle)

{

Monom a(12, 14);

Monom b(14, 43);

Monom c(11, 123);

Monom d(12, 1232);

list<Monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.Insert\_to\_tail(c);

l.Insert\_to\_tail(d);

l.Dellcurrent();

EXPECT\_EQ(l.getcurrentdate(), b);

}

TEST(list, Dellcurent\_first\_must\_empty)

{

Monom a(12, 14);

list<Monom> l;

l.Insert\_to\_tail(a);

l.reset();

l.Dellcurrent();

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

}

TEST(list, can\_get\_currnextdata)

{

Monom a(12, 14);

Monom b(14, 43);

Monom c(11, 123);

Monom d(12, 1232);

list<Monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.Insert\_to\_tail(c);

l.Insert\_to\_tail(d);

EXPECT\_EQ(l.getnextcurrentdate(), d);

}

TEST(list, can\_get\_currdata)

{

Monom a(12, 14);

Monom b(14, 43);

Monom c(11, 123);

Monom d(12, 1232);

list<Monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.Insert\_to\_tail(c);

l.Insert\_to\_tail(d);

EXPECT\_EQ(l.getcurrentdate(), c);

}

TEST(list, can\_getnext\_node)

{

Monom a(12, 14);

Monom b(14, 43);

Monom c(11, 123);

Monom d(12, 1232);

list<Monom> l;

l.Insert\_to\_tail(a);

l.Insert\_to\_tail(b);

l.Insert\_to\_tail(c);

l.Insert\_to\_tail(d);

l.getnext();

EXPECT\_EQ(l.getcurrentdate(), d);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//тесты мономов и полиномов

TEST(polinom, Assert\_trow\_to\_mult\_bug\_steepeny)

{

ASSERT\_ANY\_THROW(polinom("12+x3+x6")\*polinom("1+x7-y2"));

}

struct monom\_mod

{

string s;

bool iscorrect;

Monom from\_string;

Monom real;

monom\_mod(string s, Monom real2):from\_string(s)

{

this->s = s;

real = real2;

if (from\_string == real) iscorrect = true; else iscorrect = false;

}

};

class parameterizedmonom : public ::testing::TestWithParam<monom\_mod>

{

};

monom\_mod myarray[] = { monom\_mod("3x2",Monom(3,200)), monom\_mod("15y2",Monom(15,20)), monom\_mod("z8",Monom(1,8)), monom\_mod("-1y5x1z8",Monom(-1,158)) };

INSTANTIATE\_TEST\_CASE\_P(instantiation1,

parameterizedmonom,

::testing::ValuesIn(myarray));

TEST\_P(parameterizedmonom, can\_create\_monom\_from\_string)

{

EXPECT\_EQ(GetParam().iscorrect, true);

}

struct polinom\_mod\_pl

{

string s;

bool iscorrect\_plus;

polinom a;

polinom b;

polinom res;

polinom\_mod\_pl (string s, polinom a, polinom b) : res(s)

{

this->s = s;

this->a = a;

this->b = b;

polinom res2 = a + b;

if (res2 == res) iscorrect\_plus = true; else iscorrect\_plus = false;

}

};

class parameterizedpolinom\_plus : public ::testing::TestWithParam<polinom\_mod\_pl>

{

};

polinom\_mod\_pl myarray3[] = { polinom\_mod\_pl("x8",polinom("2x8"), polinom("-x8")),polinom\_mod\_pl("x8+y2-1",polinom("2x8+5y2+3"), polinom("-x8-4-4y2")) };

INSTANTIATE\_TEST\_CASE\_P(instantiation2,

parameterizedpolinom\_plus,

::testing::ValuesIn(myarray3));

TEST\_P(parameterizedpolinom\_plus,can\_right\_plus\_polinoms)

{

EXPECT\_EQ(GetParam().iscorrect\_plus, true);

}

struct polinom\_mod\_mul

{

string s;

bool iscorrect\_mul;

polinom a;

polinom b;

polinom res;

polinom\_mod\_mul(string s, polinom a, polinom b) : res(s)

{

this->s = s;

this->a = a;

this->b = b;

polinom res2 = a \* b;

if (res2 == res) iscorrect\_mul = true; else iscorrect\_mul = false;

}

};

class parameterizedpolinom\_multyply : public ::testing::TestWithParam<polinom\_mod\_mul>

{

};

polinom\_mod\_mul myarray4[] = { polinom\_mod\_mul("x8",polinom("2x2"), polinom("0.5x6")), polinom\_mod\_mul("z2",polinom("8z1"), polinom("0.125z1")) };

INSTANTIATE\_TEST\_CASE\_P(instantiation3,

parameterizedpolinom\_multyply,

::testing::ValuesIn(myarray4));

TEST\_P (parameterizedpolinom\_multyply, can\_right\_multyply\_polinoms)

{

EXPECT\_EQ(GetParam().iscorrect\_mul, true);

}