Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Кафедра прикладной математики и кибернетики

Лабораторная работа
«Полином»

Выполнил:
Студент группы ИП-017
Костин А.В.
Работу проверил:
ассистент кафедры ПМиК
Агалаков А.А.

Новосибирск 2023 г.

Содержание

1. Задание

1. Реализовать тип «полином», в соответствии с приведенной ниже спецификацией.
2. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования.
3. Если необходимо, предусмотрите возбуждение исключительных ситуаций.

Спецификация абстрактного типа данных «Полином».

ADT TPoly

Данные

Полиномы Tpoly - это неизменяемые полиномы с целыми коэффициентами.

Операции

Операции могут вызываться только объектом «полином» (тип TPoly), указатель на который передаётся в них по умолчанию. При описании операций этот объект в разделе «Вход» не указывается.

Спецификация абстрактного типа данных Одночлен.

ADT TMember

Данные - Одночлен TMember - это изменяемые одночленные полиномы с целыми коэффициентами. Коэффициент и степень хранятся в полях целого типа FCoeff и FDegree соответственно.

Операции - Операции могут вызываться только объектом «одночлен» (тип TMember), указатель на который передаётся в них по умолчанию. При описании операций этот объект в разделе «Вход» не указывается.

Наименование операции	Описание
Конструктор	
Начальные значения:	Коэффициент (c) и степень (n) одночленного полинома
Процесс:	Создаёт одночленный полином с коэффициентом (c) и степенью (n), или ноль-полином, если коэффициент (c) равен 0 и возвращает указатель на него. Например: Конструктор(6,3) = 6x^3 Конструктор(3,0) = 3 Конструктор() = 0
Выход:	Нет.
Постусловия:	Объект инициализирован.
Степень	

Вход:	Нет.
Предусловия:	Нет.
Процесс:	Отыскивает степень n полинома, т.е. наибольшую степень при ненулевом коэффициенте (c). Степень нулевого полинома равна 0. Например: a = (x^2+1), a.Степень = 2 a = (17), a. Степень = 0
Выход:	n - целое число - степень полинома.
Постусловия:	Нет.
Коэффициент	
Вход:	n - целое число - степень полинома.
Предусловия:	Полином – не нулевой.
Процесс:	Отыскивает коэффициент (c) при члене полинома со степенью n (c*x^n). Возвращает коэффициент (c) найденного члена или 0, если n больше степени полинома. Например: p = (x^3+2x+1), p.Coeff (4) = 0 p = (x^3+2x+1), p.Coeff (1) = 2
Выход:	Целое число.
Постусловия:	Нет.
Очистить (Clear)	
Вход:	Нет.
Предусловия:	Нет
Процесс:	Удаляет члены полинома.
Выход:	Нет.
Постусловия:	this – нуль-полином.
Сложить	
Вход:	q - полином.
Предусловия:	Нет
Процесс:	Создаёт полином, являющийся результатом сложения полинома с полиномом q и возвращает его.
Выход:	Полином.
Постусловия:	Нет.

Умножить	
Вход:	q - полином.
Предусловия:	Нет.
Процесс:	Создаёт полином, являющийся результатом умножения полинома на полином q и возвращает его.
Выход:	Полином.
Постусловия:	Нет.
Вычесть	
Вход:	q - полином.
Предусловия:	Нет.
Процесс:	Создаёт полином, являющийся результатом вычитания из полинома полинома q, и возвращает его.
Выход:	Полином.
Постусловия:	Нет.
Минус	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Создаёт полином, являющийся разностью ноль-полинома, и полинома и возвращает его.
Выход:	Полином.
Постусловия:	Нет.
Равно	
Вход:	q - полином.
Предусловия:	Нет.
Процесс:	Сравнивает полином с полиномом q на равенство. Возвращает значение True, если полиномы равны, т.е. имеют одинаковые коэффициенты при соответствующих членах, и значение False - в противном случае.
Выход:	Булевское значение.
Постусловия:	Нет.
Дифференцировать	

2. Исходный код программы

2.1. Код программы

TPolinom.h

```
#pragma once
#include <iostream>
#include <set>
#include <vector>
#include <algorithm>
#include "Member.h"

using namespace std;

class Polinom
{
private:
	set<Member> data;
public:
	Polinom();
	Polinom(int coeef, int degree);
	Polinom(Member first);
	Polinom(set<Member> newData);
	set<Member> getData();
	void add(Member newM);
	int getMaxDegree();
	int getCooef(int n);
	Member getElement(int n);
	void clear();
	Polinom operator+(Polinom a);
	Polinom operator-(Polinom a);
	Polinom operator*(Polinom a);
	Polinom diff();
	double calc(int x);
};

inline Polinom::Polinom() {}

inline Polinom::Polinom(int cooef, int degree)
{
	data.insert(Member(cooef, degree));
}

inline Polinom::Polinom(Member a)
{
```

```
	add(a);
}

inline Polinom::Polinom(set<Member> newData)
{
	for (auto i : newData)
	{
		add(i);
	}
}


inline set<Member> Polinom::getData()
{
	return data;
}

inline void Polinom::add(Member newM)
{
	Member id;
	bool isFind = false;
	for (auto i : data)
	{
		if (newM.getDegree() == i.getDegree())
		{
			id = i;
			isFind = true;
		}
	}
	if (isFind)
	{
		data.erase(id);
		data.insert(Member(id.getCooef() + newM.getCooef(),
newM.getDegree()));
	}
	else
		data.insert(newM);
}


inline string to_string(Polinom x)
{
	string res = "";
```

```
	for (auto i : x.getData())
	{
		res += to_string(i) + " + ";
	}
	if (res == "")
		return res;
	res.pop_back();
	res.pop_back();
	res.pop_back();
	return res;
}

inline int Polinom::getMaxDegree()
{
	int m = 0;
	for (auto i : data)
		m = i.getDegree();
	return m;
}

inline Member Polinom::getElement(int n)
{
	Member m(0, 0);
	int j = 0;
	for (auto i : data)
	{
		if (j == n)
		{
			m.setDegree(i.getDegree());
			m.setCooef(i.getCooef());
			break;
		}
		++j;
	}
	return m;
}

inline int Polinom::getCooef(int n)
{
	int c = -1;
	for (auto i : data)
		if (i.getDegree() == n)
			c = i.getCooef();
```

```
	return c;
}

inline void Polinom::clear()
{
	data.clear();
}

inline Polinom Polinom::operator+(Polinom a)
{
	Polinom res;
	for (auto i : a.getData())
	{
		res.add(i);
	}
	for (auto i : data)
	{
		res.add(i);
	}
	return res;
}

inline Polinom Polinom::operator-(Polinom a)
{
	Polinom res;
	for (auto i : a.getData())
	{
		res.add(Member(-i.getCooef(), i.getDegree()));
	}
	for (auto i : data)
	{
		res.add(i);
	}
	return res;
}

inline Polinom Polinom::operator*(Polinom a)
{
	Polinom res;
	for (auto i : a.getData())
	{
		for (auto j : data)
		{
```

```
                    res.add(Member(i.getCooef() * j.getCooef(), i.getDegree() +
j.getDegree()));
                }
        }
        return res;
}

inline Polinom Polinom::diff()
{
        Polinom res;
        for (auto j : data)
        {
                Member a = j.Diff();
                if (a.getCooef() == 0 and a.getDegree() == 0)
                        continue;
                res.add(a);
        }
        return res;
}

inline double Polinom::calc(int x)
{
        double res = 0;
        for (auto i : data)
        {
                res += i.calc(x);
        }
        return res;
}
```

2.2. Код тестов

UnitTest.cpp

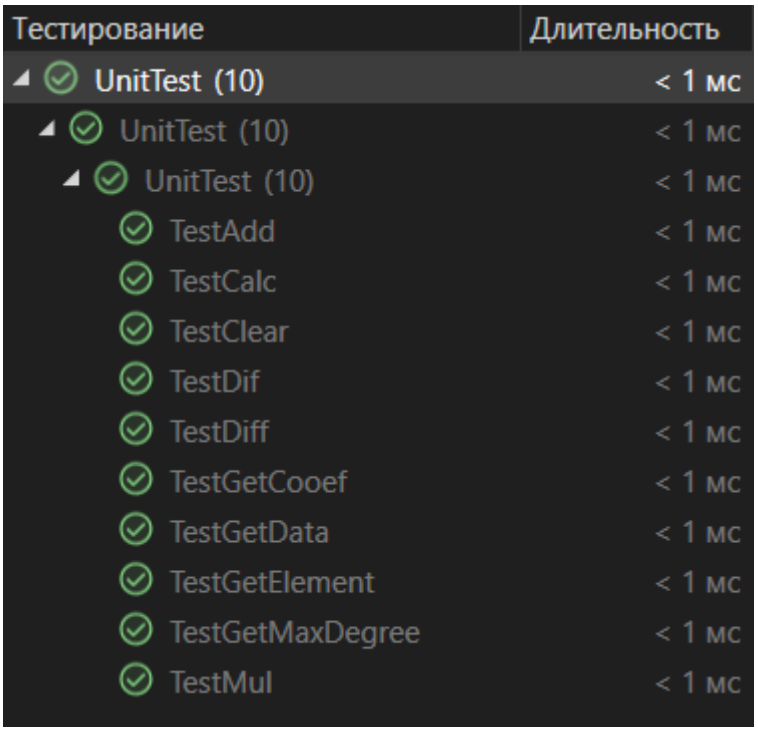
```
#include "pch.h"
#include "CppUnitTest.h"
#include "../ModernCodingPolinom/Polinom.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest
{
        TEST_CLASS(UnitTest)
```

```
{
public:

	TEST_METHOD(TestGetMaxDegree)
	{
		Polinom a;
		a.add(Member(1, 0));
		a.add(Member(2, 1));
		int res = a.getMaxDegree();
		int expected = 1;
		Assert::AreEqual(expected, res);
	}

	TEST_METHOD(TestGetCooef)
	{
		Polinom a;
		a.add(Member(1, 0));
		a.add(Member(2, 1));
		int res = a.getCooef(1);
		int expected = 2;
		Assert::AreEqual(expected, res);
	}

	TEST_METHOD(TestGetElement)
	{
		Polinom a;
		a.add(Member(1, 0));
		a.add(Member(2, 1));
		Member res = a.getElement(1);
		int expected1 = 2;
		int expected2 = 1;
		Assert::AreEqual(expected1, res.getCooef());
		Assert::AreEqual(expected2, res.getDegree());
	}

	TEST_METHOD(TestGetData)
	{
		Polinom a;
		a.add(Member(1, 0));
		a.add(Member(2, 1));
		string res = to_string(a);
		string expected = "1 + 2x";
		Assert::AreEqual(expected, res);
	}
```

```
TEST_METHOD(TestClear)
{
	Polinom a;
	a.add(Member(1, 0));
	a.add(Member(2, 1));
	a.clear();
	string res = to_string(a);
	string expected = "";
	Assert::AreEqual(expected, res);
}

TEST_METHOD(TestAdd)
{
	Polinom a(4, 1);
	a.add(Member(5, 2));
	a.add(Member(3, 0));
	Polinom b(1, 0);
	b.add(Member(2, 1));

	Polinom c;
	c = a + b;
	string res = to_string(c);
	string expected = "4 + 6x + 5x^2";
	Assert::AreEqual(expected, res);
}


TEST_METHOD(TestDif)
{
	Polinom a(4, 1);
	a.add(Member(5, 2));
	a.add(Member(3, 0));
	Polinom b(1, 0);
	b.add(Member(2, 1));

	Polinom c;
	c = a - b;
	string res = to_string(c);
	string expected = "2 + 2x + 5x^2";
	Assert::AreEqual(expected, res);
}

TEST_METHOD(TestMul)
```

```
		{
			Polinom a(4, 1);
			a.add(Member(5, 2));
			a.add(Member(3, 0));
			Polinom b(1, 0);
			b.add(Member(2, 1));

			Polinom c;
			c = a * b;
			string res = to_string(c);
			string expected = "3 + 10x + 13x^2 + 10x^3";
			Assert::AreEqual(expected, res);
		}

		TEST_METHOD(TestDiff)
		{
			Polinom a(4, 1);
			a.add(Member(5, 2));
			a.add(Member(3, 0));

			Polinom c;
			c = a.diff();
			string res = to_string(c);
			string expected = "4 + 10x";
			Assert::AreEqual(expected, res);
		}

		TEST_METHOD(TestCalc)
		{
			Polinom a(4, 1);
			a.add(Member(5, 2));
			a.add(Member(3, 0));


			double res = a.calc(2);
			double expected = 3 + 8 + 20;
			Assert::AreEqual(expected, res);
		}
	};
}
```

3. Результаты модульных тестов



4. Вывод

По итогам данной лабораторной работе были сформированы практические навыки реализации шаблона память на одно число с помощью классов C++ и их модульного тестирования.