Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет
телекоммуникаций и информатики»

Лабораторная работа №9

Выполнил: студент 4 курса
ИВТ, гр. ИП-713
Михеев Н.А.
Проверил: ассистент кафедры
ПМиК
Агалаков А.А.

Новосибирск, 2020 г.

Цель

Сформировать практические навыки реализации параметризованного абстрактного типа данных с помощью шаблона классов C++.

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.

Реализация и описание

Абстрактный тип данных «полином» был реализован посредством создания двух классов. Один класс является одночленом полинома. Другой класс – сам полином. В полях класса «одночлен» хранятся коэффициент и степень. Класс «полином» содержит единственное поле типа map<int, TMonomial> - сам полином.

- TPoly(int coeff, int degree) - создаёт одночленный полином с коэффициентом (c) и степенью (n), или ноль-полином, если коэффициент (c) равен 0 и возвращает указатель на него.
- int maxDegree() - отыскивает степень n полинома, т.е. наибольшую степень при ненулевом коэффициенте (c). Степень нулевого полинома равна 0.
- int coeff(int Degree) - отыскивает коэффициент (c) при члене полинома со степенью n (c*x^n). Возвращает коэффициент (c) найденного члена или 0, если n больше степени полинома.
- void clear() - удаляет члены полинома.

- TPoly operator+ (TPoly otherPoly) - создаёт полином, являющийся результатом сложения полинома с полиномом q и возвращает его.
- TPoly operator- (TPoly otherPoly) - создаёт полином, являющийся результатом вычитания из полинома полинома q, и возвращает его.
- TPoly operator* (TPoly otherPoly) - создаёт полином, являющийся результатом умножения полинома на полином q и возвращает его.
- TPoly minus() - создаёт полином, являющийся разностью ноль-полинома, и полинома и возвращает его.
- bool operator == (const TPoly& otherPoly) - 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.
- TPoly differentiate() - создаёт полином, являющийся производной полинома и возвращает его.
- double compute(double x) - вычисляет значение полинома в точке x и возвращает его.
- TMonomial elem(int pos) - обеспечивает доступ к члену полинома с индексом i для чтения его коэффициента (c) и степени (n) так, что если изменять i от 0 до количества членов в полиноме минус один, то можно просмотреть все члены полинома.
- TMonomial operator[] (int pos) – возвращает объект TMonomial из полинома.
- ~TPoly() – деструктор для объектов класса.

- TMonomial(int coeff, int power) – создаёт одночленный полином с коэффициентом (c) и степенью (n), или ноль-полином, если коэффициент (c) равен 0 и возвращает указатель на него.
- int readPower() – Возвращает степень n одночленного полинома (содержимое поля FDegree). Степень нулевого полинома равна 0.

- void writePower(int power) - Записывает степень n одночленного полинома в поле FDegree.
  void writeCoeff(int coeff) - Записывает коэффициент c одночленного полинома в поле FCoeff.
- TMonomial differentiate() - Создаёт одночлен, являющийся производной одночлена и возвращает его.
- double compute(double x) - Вычисляет значение одночлена в точке x и возвращает его.
- string monomialToString() - Формирует строковое представление одночлена.
- bool operator == (const TMonomial& comparable) - Сравнивает одночлен с одночленом q на равенство. Возвращает значение True, если одночлены равны, т.е. имеют одинаковые коэффициенты и степени, и значение False - в противном случае.
- ~TMonomial() – деструктор.

Заключение

В ходе данной работы согласно спецификациям задания был реализован абстрактный тип данных «полином». Также был получен практический опыт написания шаблонных функций на языке программирования C++.

Скриншоты

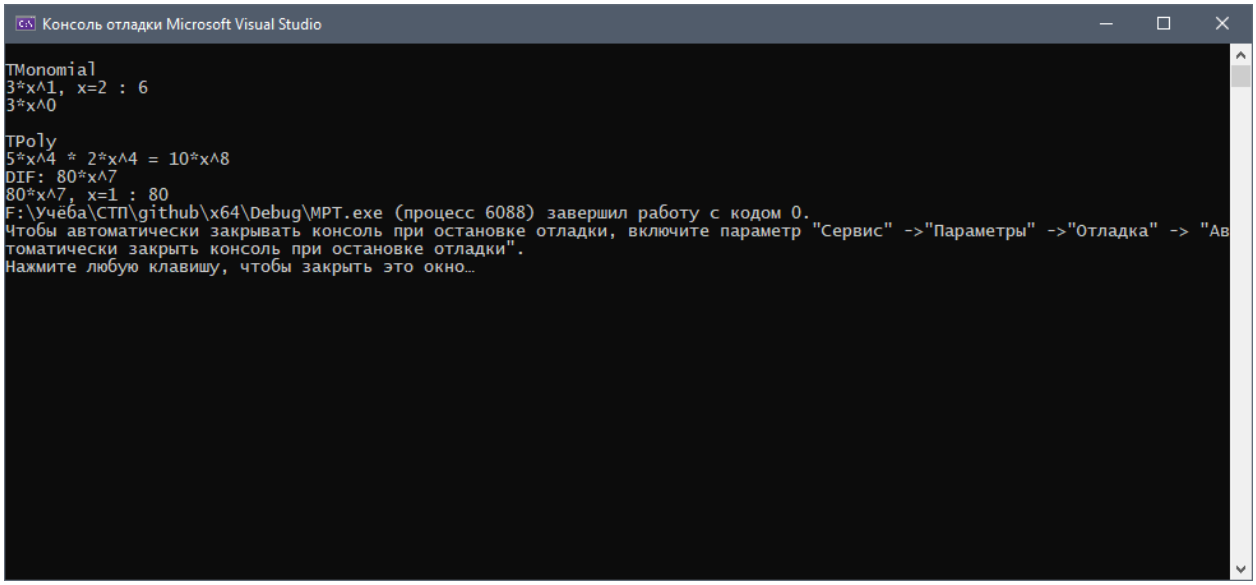

Рис. 1 – пример работы программы

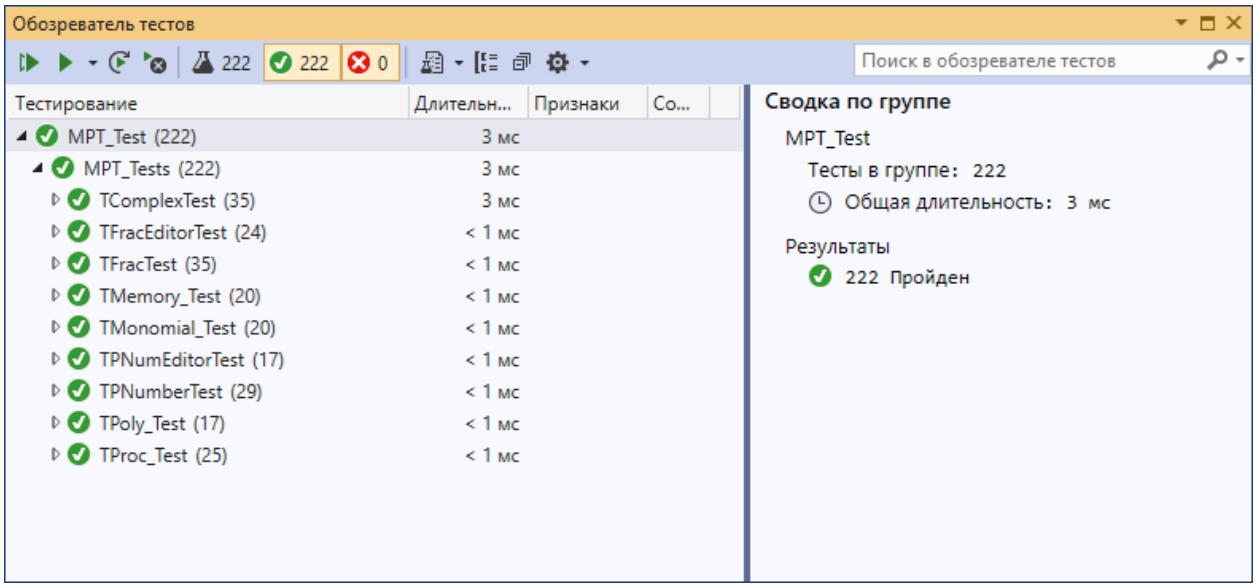

Рис. 2 – сводка проведённого тестирования программы

Код программы

Monomial.h

```
#pragma once
#include <string>
#include <stdexcept>
using namespace std;

class TMonomial {
private:
	int coeff;
	int power;
public:
	TMonomial();
	TMonomial(int coeff, int power);
	int readPower();
```

```
	void writePower(int power);
	int readCoeff();
	void writeCoeff(int coeff);
	bool isEqual(TMonomial comparable);
	TMonomial differentiate();
	double compute(double x);
	string monomialToString();
	bool operator == (const TMonomial& comparable) const {
		return this->power == comparable.power && this->coeff == comparable.coeff;
	}
	~TMonomial();
};
```

Monomial.cpp

```
#include "TMonomial.h"

TMonomial::TMonomial() {}

TMonomial::TMonomial(int coeff, int power) {
	this->coeff = coeff;
	if (coeff == 0)
	{
		this->power = 0;
	}
	else
	{
		this->power = power;
	}
}

int TMonomial::readPower() {
	return this->power;
}

void TMonomial::writePower(int power) {
	this->power = power;
}

int TMonomial::readCoeff() {
	return this->coeff;
}

void TMonomial::writeCoeff(int coeff) {
	this->coeff = coeff;
}

bool TMonomial::isEqual(TMonomial c) {
	return (this->coeff == c.coeff && this->power == c.power);
}

TMonomial TMonomial::differentiate() {
	return TMonomial(this->power == 0 ? 0 : this->coeff * this->power,
		this->power == 0 ? 0 : this->power - 1);
}

double TMonomial::compute(double x) {
	if ((this->coeff * pow(x, this->power)) > DBL_MAX) {
		throw overflow_error("Overflow");
		return (-99999);
	}
	return (this->coeff * pow(x, this->power));
}

string TMonomial::monomialToString() {
	if (this->coeff == 0)
```

```
		return string("0");
	else
		return string(to_string(this->coeff)
			+ "*x^" + to_string(this->power));
}

TMonomial::~TMonomial() {}
```

TPoly.h

```
#pragma once
#include "TMonomial.h"
#include <map>
class TPoly {
private:
	map<int, TMonomial> polynom;
public:
	TPoly();
	TPoly(int coeff, int degree);
	int maxDegree();
	int coeff(int Degree);
	void clear();
	TPoly operator+ (TPoly otherPoly);
	TPoly operator- (TPoly otherPoly);
	TPoly operator* (TPoly otherPoly);
	TPoly minus();
	bool operator == (const TPoly& otherPoly) const {
		return this->polynom == otherPoly.polynom;
	}
	TPoly differentiate();
	double compute(double x);
	TMonomial elem(int pos);
	TMonomial operator[] (int pos);
	~TPoly();
};
```

TPoly.cpp

```
#include "TPoly.h"
#include <utility>
#include <numeric>

TPoly::TPoly() {}

TPoly::TPoly(int coeff, int degree) {
	polynom.emplace(degree, TMonomial(coeff, degree));
}

int TPoly::maxDegree() {
	return polynom.rbegin()->first;
}

int TPoly::coeff(int Degree) {
	if (!polynom.count(Degree))
		return 0;
	else
		return polynom.at(Degree).readCoeff();
}

void TPoly::clear() {
	polynom.clear();
}

TPoly TPoly::operator+(TPoly otherPoly) {
	TPoly result = *this;
	for (auto& pairElem : otherPoly.polynom)
```

```
			if (result.polynom.count(pairElem.first))
				result.polynom.at(pairElem.first) =
				TMonomial(result.polynom.at(pairElem.first).readCoeff() +
pairElem.second.readCoeff(), pairElem.first);
			else
				result.polynom.emplace(pairElem);
		return result;
}

TPoly TPoly::operator*(TPoly otherPoly) {
		TPoly newPoly;
		for (auto& it1 : this->polynom)
			for (auto& it2 : otherPoly.polynom) {
				TMonomial newMember(it1.second.readCoeff() * it2.second.readCoeff(),
it1.second.readPower() + it2.second.readPower());
				if (newPoly.polynom.count(newMember.readPower()))
					newPoly.polynom.emplace(newMember.readPower(),
TMonomial(newMember.readCoeff() + newPoly.polynom.at(newMember.readPower()).readCoeff(),
newMember.readPower()));
				else
					newPoly.polynom.emplace(newMember.readPower(), newMember);
			}
		return newPoly;
}

TPoly TPoly::operator-(TPoly otherPoly) {
		TPoly result = *this;
		for (auto& pairElem : otherPoly.polynom)
			if (result.polynom.count(pairElem.first))
				result.polynom.at(pairElem.first) =
				TMonomial(result.polynom.at(pairElem.first).readCoeff() -
pairElem.second.readCoeff(), pairElem.first);
			else
				result.polynom.emplace(-pairElem.first, TMonomial(-pairElem.first,
pairElem.second.readPower()));
		return result;
}

TPoly TPoly::minus() {
		TPoly newPoly;
		for (auto& it : polynom)
			newPoly.polynom.emplace(-it.first, TMonomial(-it.second.readCoeff(),
it.second.readPower()));
		return newPoly;
}


TPoly TPoly::differentiate() {
		TPoly newPoly;
		for (auto& it : polynom)
			newPoly.polynom.emplace(it.first == 0 ? 0 : it.first - 1,
it.second.differentiate());
		return newPoly;
}

double TPoly::compute(double x) {
		if (x == 0)
			return 0;
		double sum = 0.0;
		for (auto& it : polynom)
			sum += it.second.compute(x);
		if (sum > DBL_MAX) {
			throw overflow_error("Overflow");
			return (-99999);
		}
```

```
	return sum;
}

TMonomial TPoly::elem(int pos) {
	if (pos < 0 || pos >(int)polynom.size())
		return TMonomial();
	else {
		int cntr = 0;
		for (auto& it : polynom)
			if (cntr == pos)
				return it.second;
	}
	return TMonomial();
}

TMonomial TPoly::operator[](int pos) {
	if (pos < 0 || pos > (int)polynom.size())
		return TMonomial();
	else {
		int cntr = 0;
		for (auto& it : polynom)
			if (cntr++ == pos)
				return it.second;
	}
	return TMonomial();
}

TPoly::~TPoly() {}
```