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

«Класс Полином»

Выполнил:

Студент группы ИП-113

Шпилев Д. И.

Работу проверил:

старший преподаватель кафедры ПМиК

Агалаков А.А.

Новосибирск 2024 г.

**Содержание**

[**1.** **Задание** 3](#_Toc177751483)

[**2.1 Код программы** 3](#_Toc177751484)

[**2.2** **Код тестов** 7](#_Toc177751485)

[**3.** **Результаты модульных тестов** 9](#_Toc177751486)

[**4.** **Вывод** 9](#_Toc177751487)

1. **Задание**

1. Реализовать тип «полином», в соответствии с приведенной ниже спецификацией.

2. Протестировать каждую операцию, определенную на типе данных, используя средства модульного тестирования Visual Studio.

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.

**2.1** **Код программы**

**TMember.h**

#pragma once

#include <iostream>

#include <string>

class TMember

{

public:

TMember(long long coeff, unsigned long long degree) : FCoeff(coeff), FDegree(degree)

{

if (FCoeff == 0) {

FDegree = 0;

}

}

unsigned long long getDegree() const noexcept { return FDegree; }

void setDegree(unsigned long long degree) { FDegree = degree; }

long long getCoeff() const noexcept { return FCoeff; }

void setCoeff(long long coeff);

TMember differentiate() const noexcept;

long long calculate(int x) const noexcept;

std::string toString() const noexcept;

bool operator ==(const TMember& other) const noexcept { return FCoeff == other.FCoeff && FDegree == other.FDegree; }

bool operator<(const TMember& other) const noexcept { return FDegree > other.FDegree; }

private:

long long FCoeff;

unsigned long long FDegree;

};

inline void TMember::setCoeff(long long coeff) {

FCoeff = coeff;

if (FCoeff == 0)

FDegree = 0;

}

inline TMember TMember::differentiate() const noexcept

{

if (FDegree == 0)

{

return TMember(0, 0);

}

return TMember(FCoeff \* FDegree, FDegree - 1);

}

inline long long TMember::calculate(int x) const noexcept

{

return FCoeff \* pow(x, FDegree);

}

inline std::string TMember::toString() const noexcept

{

std::string str;

if (FCoeff != 0) {

str = std::to\_string(FCoeff);

if (FDegree > 0) {

str += "x";

if (FDegree > 1) {

str += "^" + std::to\_string(FDegree);

}

}

}

else {

str = "0";

}

return str;

}

**TPoly.h**

#pragma once

#include <set>

#include "TMember.h"

class TPoly

{

public:

TPoly() {}

TPoly(long long coeff, unsigned long long degree) { polynom.insert(TMember(coeff, degree)); }

unsigned long long degree() const noexcept;

long long coeff(unsigned long long degree) const noexcept;

void addMember(long long coeff, unsigned long long degree);

void clear();

TPoly add(const TPoly& other) const noexcept;

TPoly multiply(const TPoly& other) const noexcept;

TPoly subtract(const TPoly& other) const noexcept;

TPoly negate() const noexcept;

TPoly differentiate() const noexcept;

long long calculate(long long x) const noexcept;

TMember at(size\_t index) const;

bool operator==(const TPoly& other) const noexcept { return polynom == other.polynom; }

private:

std::set<TMember> polynom;

};

**TPoly.cpp**

#include "TPoly.h"

unsigned long long TPoly::degree() const noexcept

{

if (polynom.empty()) {

return 0;

}

return polynom.begin()->getDegree();

}

long long TPoly::coeff(unsigned long long degree) const noexcept

{

for (auto& it : polynom) {

if (it.getDegree() == degree) {

return it.getCoeff();

}

}

return 0;

}

void TPoly::addMember(long long coeff, unsigned long long degree)

{

if (coeff == 0) {

return;

}

auto it = polynom.find(TMember(coeff, degree));

if (it != polynom.end()) {

int newCoeff = it->getCoeff() + coeff;

polynom.erase(it);

if (newCoeff != 0) {

polynom.insert(TMember(newCoeff, degree));

}

}

else {

polynom.insert(TMember(coeff, degree));

}

}

void TPoly::clear()

{

polynom.clear();

}

TPoly TPoly::add(const TPoly& other) const noexcept

{

TPoly result = \*this;

for (const auto& member : other.polynom) {

result.addMember(member.getCoeff(), member.getDegree());

}

return result;

}

TPoly TPoly::multiply(const TPoly& other) const noexcept

{

TPoly result;

for (const auto& member1 : polynom) {

for (const auto& member2 : other.polynom) {

result.addMember(member1.getCoeff() \* member2.getCoeff(), member1.getDegree() + member2.getDegree());

}

}

return result;

}

TPoly TPoly::subtract(const TPoly& other) const noexcept

{

TPoly result = \*this;

for (const auto& member : other.polynom) {

result.addMember(-member.getCoeff(), member.getDegree());

}

return result;

}

TPoly TPoly::negate() const noexcept

{

TPoly result;

for (const auto& member : polynom) {

result.addMember(-member.getCoeff(), member.getDegree());

}

return result;

}

TPoly TPoly::differentiate() const noexcept

{

TPoly result;

for (const auto& member : polynom) {

TMember diffMember = member.differentiate();

if (diffMember.getCoeff() != 0) {

result.addMember(diffMember.getCoeff(), diffMember.getDegree());

}

}

return result;

}

long long TPoly::calculate(long long x) const noexcept

{

double result = 0;

for (const auto& member : polynom) {

result += member.calculate(x);

}

return result;

}

TMember TPoly::at(size\_t index) const

{

if (index >= polynom.size()) {

throw std::out\_of\_range("Index " + std::to\_string(index) + " out of range");

}

auto it = polynom.begin();

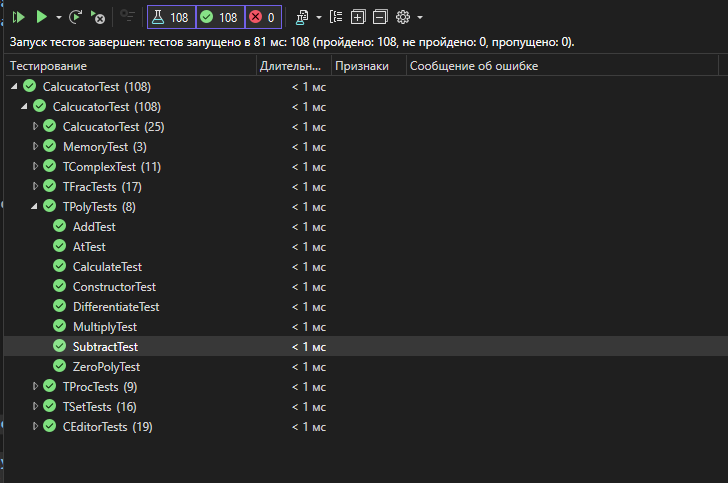
std::advance(it, index);

return \*it;

}

* 1. **Код тестов**

1. TEST\_CLASS(TPolyTests)
2. {
3. public:
4. TEST\_METHOD(ConstructorTest)
5. {
6. TPoly poly(5, 3);
7. unsigned long long res = poly.degree();
8. auto t = poly.coeff(3);
9. Assert::AreEqual(3ull, poly.degree());
10. Assert::AreEqual(5ll, poly.coeff(3));
11. }
12. TEST\_METHOD(ZeroPolyTest)
13. {
14. TPoly poly(0, 0);
15. Assert::AreEqual(0ull, poly.degree());
16. Assert::AreEqual(0ll, poly.coeff(0));
17. }
18. TEST\_METHOD(AddTest)
19. {
20. TPoly poly1(3, 2);
21. TPoly poly2(5, 2);
22. TPoly result = poly1.add(poly2);
23. Assert::AreEqual(2ull, result.degree());
24. Assert::AreEqual(8ll, result.coeff(2));
25. }
26. TEST\_METHOD(MultiplyTest)
27. {
28. TPoly poly1(2, 1);
29. TPoly poly2(3, 2);
30. TPoly result = poly1.multiply(poly2);
31. Assert::AreEqual(3ull, result.degree());
32. Assert::AreEqual(6ll, result.coeff(3));
33. }
34. TEST\_METHOD(SubtractTest)
35. {
36. TPoly poly1(7, 3);
37. TPoly poly2(2, 3);
38. TPoly result = poly1.subtract(poly2);
39. Assert::AreEqual(3ull, result.degree());
40. Assert::AreEqual(5ll, result.coeff(3));
41. }
42. TEST\_METHOD(DifferentiateTest)
43. {
44. TPoly poly(4, 3);
45. TPoly result = poly.differentiate();
46. Assert::AreEqual(2ull, result.degree());
47. Assert::AreEqual(12ll, result.coeff(2));
48. }
49. TEST\_METHOD(CalculateTest)
50. {
51. TPoly poly(2, 2);
52. long long result = poly.calculate(3);
53. Assert::AreEqual(18ll, result);
54. }
55. TEST\_METHOD(AtTest)
56. {
57. TPoly poly;
58. poly.addMember(6, 4);
59. TMember member = poly.at(0);
60. Assert::AreEqual(6ll, member.getCoeff());
61. Assert::AreEqual(4ull, member.getDegree());
62. }
63. };
64. **Результаты модульных тестов**

****

1. **Вывод**

По итогам данной лабораторной работе были сформированы практические навыки реализации абстрактных типов данных в соответствии с заданной спецификацией с помощью классов С++ и их модульного тестирования.