

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
2.1 Код программы	3
2.2 Код тестов.....	7
3. Результаты модульных тестов	9
4. Вывод	9

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 TPolу

Данные

Полиномы TPolу - это неизменяемые полиномы с целыми коэффициентами. Операции могут вызываться только объектом «полином» (тип TPolу), 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 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;

```

```

    TPolynomial subtract(const TPolynomial& other) const noexcept;
    TPolynomial negate() const noexcept;
    TPolynomial differentiate() const noexcept;
    long long calculate(long long x) const noexcept;
    TMember at(size_t index) const;

    bool operator==(const TPolynomial& other) const noexcept { return polynom ==
other.polynom; }
private:
    std::set<TMember> polynom;
};

```

TPolynomial.cpp

```

#include "TPolynomial.h"

unsigned long long TPolynomial::degree() const noexcept
{
    if (polynom.empty()) {
        return 0;
    }
    return polynom.begin()->getDegree();
}

long long TPolynomial::coeff(unsigned long long degree) const noexcept
{
    for (auto& it : polynom) {
        if (it.getDegree() == degree) {
            return it.getCoeff();
        }
    }
    return 0;
}

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

TPolynomial TPolynomial::add(const TPolynomial& other) const noexcept
{
    TPolynomial result = *this;
    for (const auto& member : other.polynom) {
        result.addMember(member.getCoeff(), member.getDegree());
    }
    return result;
}

```

```

}

TPoly TPolynomial::multiply(const TPolynomial& other) const noexcept
{
    TPolynomial 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 TPolynomial::subtract(const TPolynomial& other) const noexcept
{
    TPolynomial result = *this;
    for (const auto& member : other.polynom) {
        result.addMember(-member.getCoeff(), member.getDegree());
    }
    return result;
}

TPoly TPolynomial::negate() const noexcept
{
    TPolynomial result;
    for (const auto& member : polynom) {
        result.addMember(-member.getCoeff(), member.getDegree());
    }
    return result;
}

TPoly TPolynomial::differentiate() const noexcept
{
    TPolynomial result;
    for (const auto& member : polynom) {
        TMember diffMember = member.differentiate();
        if (diffMember.getCoeff() != 0) {
            result.addMember(diffMember.getCoeff(), diffMember.getDegree());
        }
    }
    return result;
}

long long TPolynomial::calculate(long long x) const noexcept
{
    double result = 0;
    for (const auto& member : polynom) {
        result += member.calculate(x);
    }
    return result;
}

TMember TPolynomial::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;
}

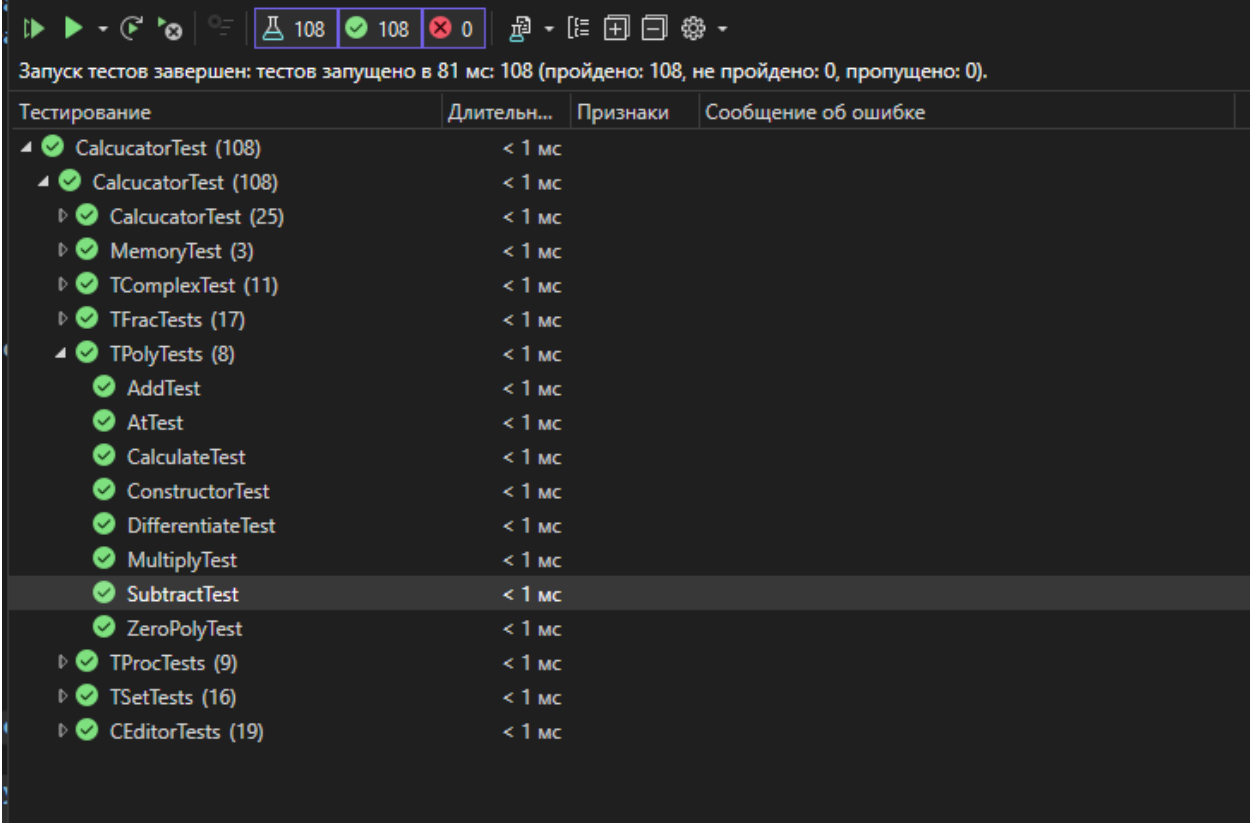
```

2.2 Код тестов

```
3  TEST_CLASS(TPolyTests)
4  {
5  public:
6
7      TEST_METHOD(ConstructorTest)
8      {
9          TPoly poly(5, 3);
10         unsigned long long res = poly.degree();
11         auto t = poly.coeff(3);
12         Assert::AreEqual(3ull, poly.degree());
13         Assert::AreEqual(5ll, poly.coeff(3));
14     }
15
16     TEST_METHOD(ZeroPolyTest)
17     {
18         TPoly poly(0, 0);
19         Assert::AreEqual(0ull, poly.degree());
20         Assert::AreEqual(0ll, poly.coeff(0));
21     }
22
23     TEST_METHOD(AddTest)
24     {
25         TPoly poly1(3, 2);
26         TPoly poly2(5, 2);
27         TPoly result = poly1.add(poly2);
28         Assert::AreEqual(2ull, result.degree());
29         Assert::AreEqual(8ll, result.coeff(2));
30     }
31
32     TEST_METHOD(MultiplyTest)
33     {
34         TPoly poly1(2, 1);
35         TPoly poly2(3, 2);
36         TPoly result = poly1.multiply(poly2);
37         Assert::AreEqual(3ull, result.degree());
38         Assert::AreEqual(6ll, result.coeff(3));
39     }
40
41     TEST_METHOD(SubtractTest)
42     {
43         TPoly poly1(7, 3);
44         TPoly poly2(2, 3);
45         TPoly result = poly1.subtract(poly2);
46         Assert::AreEqual(3ull, result.degree());
47         Assert::AreEqual(5ll, result.coeff(3));
48     }
49
50     TEST_METHOD(DifferentiateTest)
51     {
52         TPoly poly(4, 3);
53         TPoly result = poly.differentiate();
54         Assert::AreEqual(2ull, result.degree());
55         Assert::AreEqual(12ll, result.coeff(2));
56     }
57
58     TEST_METHOD(CalculateTest)
59     {
60         TPoly poly(2, 2);
61         long long result = poly.calculate(3);
62         Assert::AreEqual(18ll, result);
63     }
64
65     TEST_METHOD(AtTest)
66     {
```

```
67         TPoly poly;
68         poly.addMember(6, 4);
69         TMember member = poly.at(0);
70         Assert::AreEqual(6ll, member.getCoeff());
71         Assert::AreEqual(4ull, member.getDegree());
72     }
73 };
```


3. Результаты модульных тестов



Запуск тестов завершен: тестов запущено в 81 мс: 108 (пройдено: 108, не пройдено: 0, пропущено: 0).

Тестирование	Длительн...	Признаки	Сообщение об ошибке
✓ CalcucatorTest (108)	< 1 мс		
✓ CalcucatorTest (108)	< 1 мс		
✓ CalcucatorTest (25)	< 1 мс		
✓ MemoryTest (3)	< 1 мс		
✓ TComplexTest (11)	< 1 мс		
✓ TFracTests (17)	< 1 мс		
✓ TPolyTests (8)	< 1 мс		
✓ AddTest	< 1 мс		
✓ AtTest	< 1 мс		
✓ CalculateTest	< 1 мс		
✓ ConstructorTest	< 1 мс		
✓ DifferentiateTest	< 1 мс		
✓ MultiplyTest	< 1 мс		
✓ SubtractTest	< 1 мс		
✓ ZeroPolyTest	< 1 мс		
✓ TProcTests (9)	< 1 мс		
✓ TSetTests (16)	< 1 мс		
✓ CEditorTests (19)	< 1 мс		

4. Вывод

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