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

«Шаблон класса память на одно число»

Выполнил:

Студент группы ИП-113

Шпилев Д. И.

Работу проверил:

старший преподаватель кафедры ПМиК

Агалаков А.А.

Новосибирск 2024 г.

**Содержание**

[**1.** **Задание** 3](#_Toc177578058)

[**2.** **Исходный код программы** 3](#_Toc177578059)

[**2.1.** **Код программы** 3](#_Toc177578060)

[**2.2** **Код тестов** **Ошибка! Закладка не определена.**](#_Toc177578061)

[**3.** **Результаты модульных тестов** 8](#_Toc177578062)

[**4.** **Вывод** 8](#_Toc177578063)

1. **Задание**

1. В соответствии с приведенной ниже спецификацией реализовать параметризованный абстрактный тип данных «Процессор», используя шаблон классов С++.

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

3. Если необходимо, предусмотрите возбуждение исключительных ситуаций. Спецификация типа данных «Процессор».

**ADT TProc**

Данные Процессор (тип TProc) выполняет двухоперандные операции TOprtn = (None, Add, Sub, Mul, Dvd) и однооперандные операции - функции TFunc = (Rev, Sqr) над значениями типа T. Левый операнд и результат операции хранится в поле Lop\_Res, правый - в поле Rop. Оба поля имеют тип T. Процессор может находиться в состояниях: «операция установлена» - поле Operation не равно None (значение типа TOprtn) или в состоянии «операция не установлена» - поле Operation = None. Значения типа TProc - изменяемые. Они изменяются операциями: «Сброс операции» (OprtnClear), «Выполнить операцию» (OprtnRun), «Вычислить функцию» (FuncRun), «Установить операцию» (OprtnSet), «Установить левый операнд» (Lop\_Res\_Set), «Установить правый операнд» (Rop\_Set), «Сброс калькулятора» (ReSet). На значениях типа T должны быть определены указанные выше операции и функции.

* 1. **Код программы**

**UProcssr.h**

#pragma once

#include "UANumber.h"

template <class T>

class TProc

{

public:

enum Operation {

None,

Add,

Sub,

Mul,

Div

};

enum Function {

Sqr,

Rev

};

TProc();

void resetProc();

void resetOper() { m\_operation = None; }

void doOperation();

void doFunction(Function function);

std::unique\_ptr<TANumber> getLeftOperand() const noexcept { return m\_leftOperand->Clone(); }

std::unique\_ptr<TANumber> getRightOperand() const noexcept { return m\_rightOperand->Clone(); }

void setLeftOperand(const std::unique\_ptr<TANumber>& number);

void setRightOperand(const std::unique\_ptr<TANumber>& number);

Operation getOperation() const noexcept { return m\_operation; }

void setOperation(Operation operation) noexcept { m\_operation = operation; }

private:

std::unique\_ptr<TANumber> m\_leftOperand;

std::unique\_ptr<TANumber> m\_rightOperand;

Operation m\_operation;

};

template<class T>

inline TProc<T>::TProc()

{

resetProc();

}

template<class T>

inline void TProc<T>::resetProc()

{

m\_leftOperand = std::make\_unique<T>();

m\_rightOperand = std::make\_unique<T>();

m\_operation = None;

}

template<class T>

inline void TProc<T>::doOperation()

{

try {

switch (m\_operation)

{

case Add:

m\_leftOperand = \*m\_leftOperand + \*m\_rightOperand;

break;

case Sub:

m\_leftOperand = \*m\_leftOperand - \*m\_rightOperand;

break;

case Mul:

m\_leftOperand = \*m\_leftOperand \* \*m\_rightOperand;

break;

case Div:

m\_leftOperand = \*m\_leftOperand / \*m\_rightOperand;

break;

default:

break;

}

}

catch (DivisionByZeroException& ex)

{

}

catch (DifferentBaseOrPrecision& ex)

{

}

//TODO: сделать нормальное перенаправление в ERROR

}

template<class T>

inline void TProc<T>::doFunction(Function function)

{

try {

switch (function)

{

case Sqr:

m\_rightOperand = m\_rightOperand->Square();

break;

case Rev:

m\_rightOperand = m\_rightOperand->Invert();

break;

default:

break;

}

}

catch (DivisionByZeroException& ex)

{

}

catch (DifferentBaseOrPrecision& ex)

{

}

//TODO: сделать нормальное перенаправление в ERROR

}

template<class T>

inline void TProc<T>::setLeftOperand(const std::unique\_ptr<TANumber>& number)

{

const T\* pB = dynamic\_cast<const T\*>(number.get());

if (!pB)

{

throw TypeMismatchException();

}

m\_leftOperand = number->Clone();

}

template<class T>

inline void TProc<T>::setRightOperand(const std::unique\_ptr<TANumber>& number)

{

const T\* pB = dynamic\_cast<const T\*>(number.get());

if (!pB)

{

throw TypeMismatchException();

}

m\_rightOperand = number->Clone();

}

* 1. **Результаты модульных тестов**

TEST\_CLASS(TProcTests)

{

public:

TEST\_METHOD(TestResetProc)

{

TProc<TComplex> processor;

processor.resetProc();

Assert::IsTrue(processor.getLeftOperand() != nullptr);

Assert::IsTrue(processor.getRightOperand() != nullptr);

Assert::AreEqual(static\_cast<int>(TProc<TComplex>::None), static\_cast<int>(processor.getOperation()));

}

TEST\_METHOD(TestSetLeftOperand)

{

TProc<TComplex> processor;

std::unique\_ptr<TANumber> complexNumber = std::make\_unique<TComplex>(1, 1);

processor.setLeftOperand(complexNumber);

Assert::AreEqual(complexNumber->numberString(), processor.getLeftOperand()->numberString());

}

TEST\_METHOD(TestSetRightOperand)

{

TProc<TComplex> processor;

std::unique\_ptr<TANumber> complexNumber = std::make\_unique<TComplex>(2, 2);

processor.setRightOperand(complexNumber);

Assert::AreEqual(complexNumber->numberString(), processor.getRightOperand()->numberString());

}

TEST\_METHOD(TestAdditionOperation)

{

TProc<TComplex> processor;

std::unique\_ptr<TANumber> leftNumber = std::make\_unique<TComplex>(1, 1);

std::unique\_ptr<TANumber> rightNumber = std::make\_unique<TComplex>(2, 2);

processor.setLeftOperand(leftNumber);

processor.setRightOperand(rightNumber);

processor.setOperation(TProc<TComplex>::Add);

processor.doOperation();

Assert::AreEqual(TComplex(3, 3).numberString(), processor.getLeftOperand()->numberString());

}

TEST\_METHOD(TestSubtractionOperation)

{

TProc<TComplex> processor;

std::unique\_ptr<TANumber> leftNumber = std::make\_unique<TComplex>(5, 5);

std::unique\_ptr<TANumber> rightNumber = std::make\_unique<TComplex>(2, 2);

processor.setLeftOperand(leftNumber);

processor.setRightOperand(rightNumber);

processor.setOperation(TProc<TComplex>::Sub);

processor.doOperation();

Assert::AreEqual(TComplex(3, 3).numberString(), processor.getLeftOperand()->numberString());

}

TEST\_METHOD(TestMultiplicationOperation)

{

TProc<TComplex> processor;

std::unique\_ptr<TANumber> leftNumber = std::make\_unique<TComplex>(1, 1);

std::unique\_ptr<TANumber> rightNumber = std::make\_unique<TComplex>(2, 2);

processor.setLeftOperand(leftNumber);

processor.setRightOperand(rightNumber);

processor.setOperation(TProc<TComplex>::Mul);

processor.doOperation();

Assert::AreEqual(TComplex(0, 4).numberString(), processor.getLeftOperand()->numberString());

}

TEST\_METHOD(TestDivisionOperation)

{

TProc<TComplex> processor;

std::unique\_ptr<TANumber> leftNumber = std::make\_unique<TComplex>(1, 1);

std::unique\_ptr<TANumber> rightNumber = std::make\_unique<TComplex>(1, -1);

processor.setLeftOperand(leftNumber);

processor.setRightOperand(rightNumber);

processor.setOperation(TProc<TComplex>::Div);

processor.doOperation();

Assert::AreEqual(TComplex(0, 1).numberString(), processor.getLeftOperand()->numberString());

}

TEST\_METHOD(TestSquareFunction)

{

TProc<TComplex> processor;

std::unique\_ptr<TANumber> rightNumber = std::make\_unique<TComplex>(2, 3);

processor.setRightOperand(rightNumber);

processor.doFunction(TProc<TComplex>::Sqr);

Assert::AreEqual(TComplex(-5, 12).numberString(), processor.getRightOperand()->numberString());

}

TEST\_METHOD(TestInvertFunction)

{

TProc<TComplex> processor;

std::unique\_ptr<TANumber> rightNumber = std::make\_unique<TComplex>(2, 3);

processor.setRightOperand(rightNumber);

processor.doFunction(TProc<TComplex>::Rev);

Assert::AreEqual(TComplex(0.153846, -0.230769).numberString(), processor.getRightOperand()->numberString());

}

};

1. **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.