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

Лабораторная работа №10

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 абстрактных типов данных с помощью классов C++, шаблонов и библиотеки шаблонов STL, ассоциативного контейнера set.

Задание

1. В соответствии с приведенной ниже спецификацией реализуйте шаблон классов «множество». Для тестирования в качестве параметра шаблона T выберите типы:
- int;
- TFrac (простая дробь), разработанный вами ранее.
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 «множество» был реализован посредством создания шаблона класса. Класс «множество» содержит единственное поле типа set<T> - само множество.

- TSet() – Создаёт пустое множество элементов типа T.
- void clear() – Удаляет из множества все элементы.
- void insert_(T a) – Добавляет d во множество, если в нем нет такого элемента.
- void del(T a) – Удаляет элемент d из множества, если d принадлежит множеству.
- bool isEmpty() – Определяет, содержит ли множество элементы. Возвращает значение True, если множество не пусто, False – в противном случае.

- bool contains(T a) – Определяет, принадлежит ли элемент d множеству. Возвращает True, если d принадлежит множеству, False – в противном случае.
- TSet<T> add(const TSet<T>& otherSet) – Создаёт множество, полученное в результате объединения множества с множеством q.
- TSet<T> subtract(TSet<T> set) – Создаёт множество, полученное в результате вычитания из множества множество q.
- TSet<T> multiply(TSet<T> set) – Создаёт множество, являющееся пересечением множества с множеством q.
- int count() – Подсчитывает и возвращает количество элементов во множестве, если множество пустое - ноль
- T element(int num) – Обеспечивает доступ к элементу множества для чтения по индексу j так, что если изменять j от 0 до количества элементов во множестве, то можно просмотреть все элементы множества.
- ~TSet() – деструктор объектов.

Заключение

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

Скриншоты

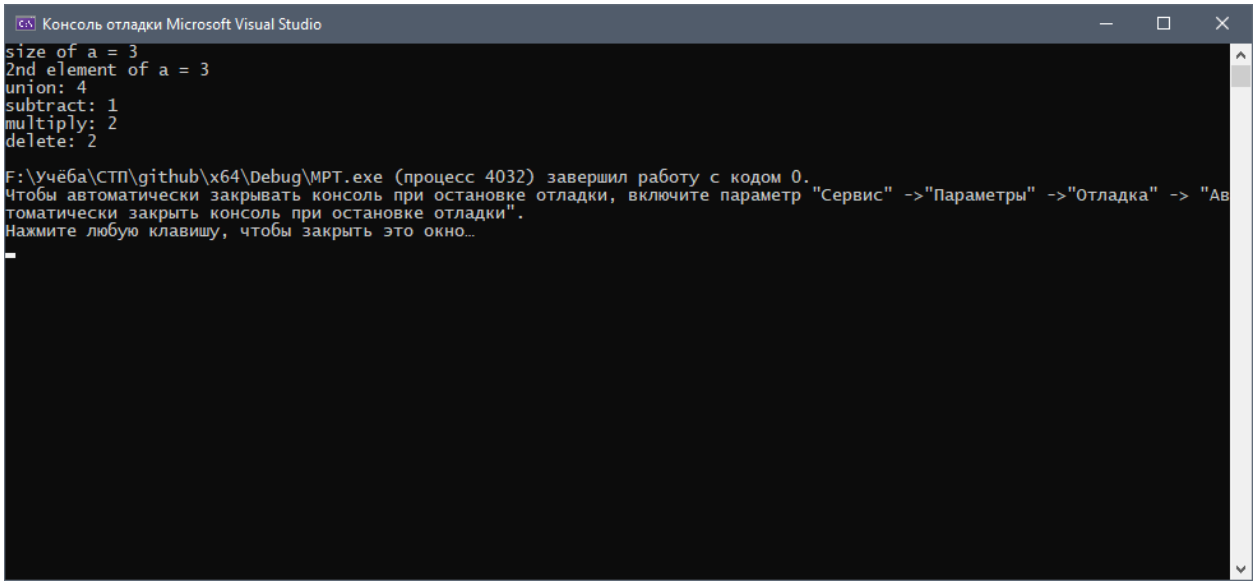

Рис. 1 – пример работы программы

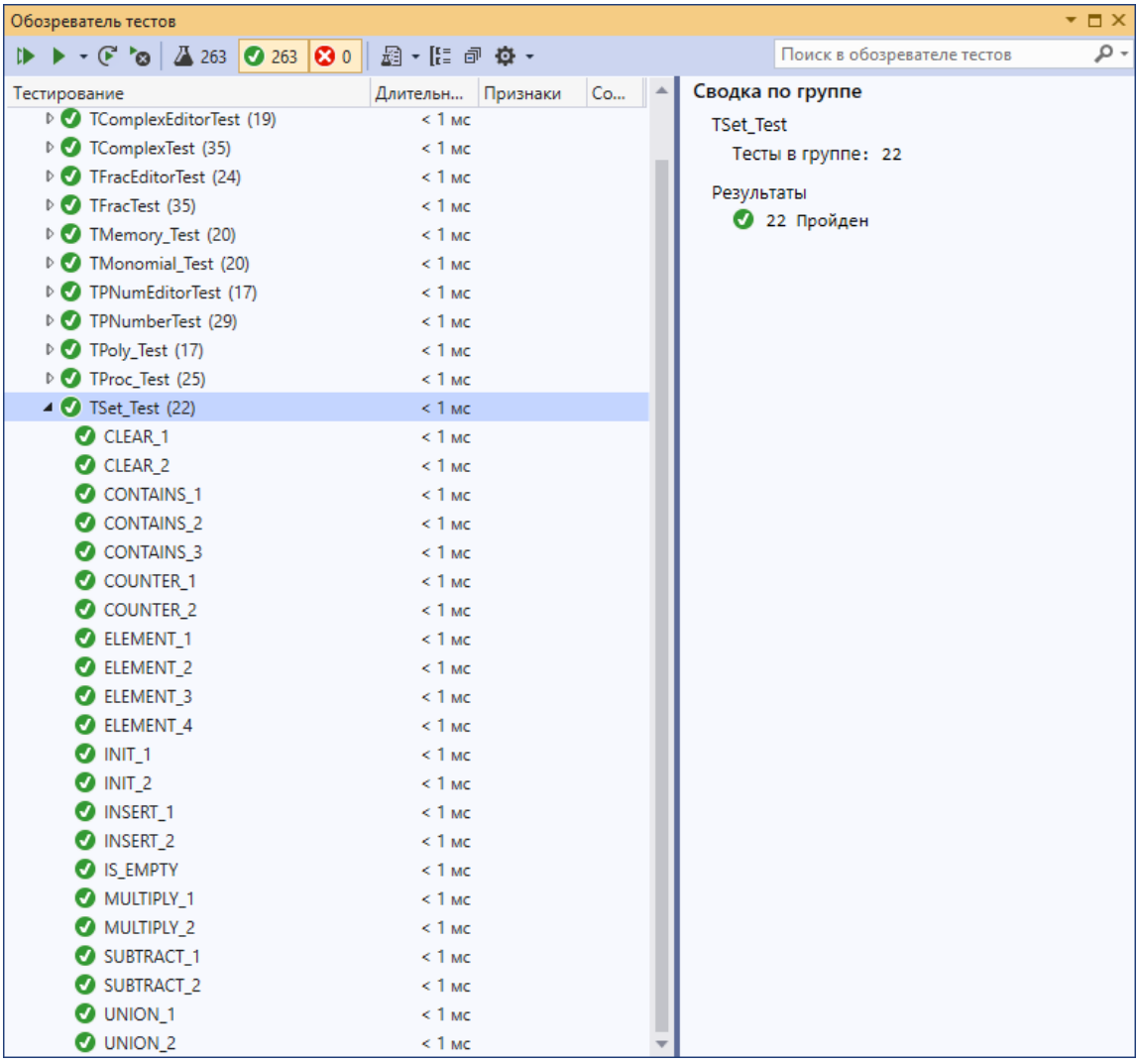

Рис. 2 – сводка проведённого тестирования программы

Код программы

TSet.h

```
#pragma once
#include <set>
#include <algorithm>
using namespace std;

template <class T>

class TSet {

public:
	set<T> container;
	TSet();
	void clear();
	void insert_(T a);
	void del(T a);
	bool isEmpty();
	bool contains(T a);
	TSet<T> add(const TSet<T>& otherSet);
	TSet<T> subtract(TSet<T> set);
	TSet<T> multiply(TSet<T> set);
	int count();
	T element(int num);
	~TSet();
};

template<class T>
TSet<T>::TSet() {}

template<class T>
void TSet<T>::clear() {
	container.clear();
}

template<class T>
void TSet<T>::insert_(T a) {
	container.insert(a);
}

template<class T>
void TSet<T>::del(T a) {
	if (container.find(a) != container.end())
		container.erase(a);
}

template<class T>
bool TSet<T>::isEmpty() {
	return container.empty();
}

template<class T>
bool TSet<T>::contains(T a) {
	return container.find(a) != container.end();
}

template<class T>
TSet<T> TSet<T>::add(const TSet<T>& otherSet) {
	TSet<T> result = *this;
	for (const T& a : otherSet.container)
		result.insert_(a);
	return result;
}

template<class T>
```

```
TSet<T> TSet<T>::subtract(TSet<T> otherSet) {
	TSet<T> result = *this;
	for (const T& a : otherSet.container)
		if (result.container.find(a) != result.container.end())
			result.del(a);
	return result;
}

template<class T>
TSet<T> TSet<T>::multiply(TSet<T> otherSet) {
	TSet<T> result;
	for (const T& a : otherSet.container)
		if (container.count(a))
			result.insert_(a);
	return result;
}

template<class T>
int TSet<T>::count() {
	return container.size();
}

template<class T>
T TSet<T>::element(int num) {
	if (num >=0 && num < container.size())
		return *next(container.begin(), num);
	throw invalid_argument("Invalid type");
	T abc;
	return abc;
}

template<class T>
TSet<T>::~TSet() {}
```

TSet_Test.cpp

```
#include "pch.h"
#include "CppUnitTest.h"
#include <set>

#include "../MPT/TSet.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace MPT_Tests
{
	TEST_CLASS(TSet_Test)
	{
	public:
		TEST_METHOD(INIT_1) {
			TSet<int> a;
			set<int> b{};
			Assert::AreEqual(true, a.container == b);
		}

		TEST_METHOD(INIT_2) {
			TSet<int> a;
			set<int> b{1};
			Assert::AreEqual(true, a.container != b);
		}

		TEST_METHOD(CLEAR_1) {
			TSet<int> a;
			a.insert_(1);
			a.clear();
			set<int> b{};
```

```
	Assert::AreEqual(true, a.container == b);
}

TEST_METHOD(CLEAR_2) {
	TSet<int> a;
	a.insert_(1);
	a.insert_(2);
	a.clear();
	set<int> b{};
	Assert::AreEqual(true, a.container == b);
}

TEST_METHOD(INSERT_1) {
	TSet<int> a;
	a.insert_(1);
	a.insert_(2);
	set<int> b{1, 2};
	Assert::AreEqual(true, a.container == b);
}

TEST_METHOD(INSERT_2) {
	TSet<int> a;
	a.insert_(1);
	a.insert_(2);
	a.insert_(3);
	set<int> b{ 1, 2, 3 };
	Assert::AreEqual(true, a.container == b);
}

TEST_METHOD(IS_EMPTY) {
	TSet<int> a;
	Assert::AreEqual(true, a.count() == 0);
}

TEST_METHOD(CONTAINS_1) {
	TSet<int> a;
	a.insert_(1);
	a.insert_(2);
	a.insert_(3);
	Assert::AreEqual(true, a.contains(1));
}

TEST_METHOD(CONTAINS_2) {
	TSet<int> a;
	a.insert_(1);
	a.insert_(2);
	a.insert_(3);
	Assert::AreEqual(true, a.contains(2));
}

TEST_METHOD(CONTAINS_3) {
	TSet<int> a;
	a.insert_(1);
	a.insert_(2);
	a.insert_(3);
	Assert::AreEqual(true, a.contains(3));
}

TEST_METHOD(UNION_1) {
	TSet<int> a;
	TSet<int> b;
	set<int> c { 1, 2, 3, 4 };
	a.insert_(1);
	a.insert_(2);
	a.insert_(3);
```

```
		b.insert_(4);
		TSet<int> result = a.add(b);
		Assert::AreEqual(true, result.container == c);
	}

	TEST_METHOD(UNION_2) {
		TSet<int> a;
		TSet<int> b;
		set<int> c{ 1, 2, 3 };
		a.insert_(1);
		a.insert_(2);
		a.insert_(3);
		b.insert_(1);
		b.insert_(2);
		b.insert_(3);
		TSet<int> result = a.add(b);
		Assert::AreEqual(true, result.container == c);
	}

	TEST_METHOD(SUBTRACT_1) {
		TSet<int> a;
		TSet<int> b;
		set<int> c{ 2, 3 };
		a.insert_(1);
		a.insert_(2);
		a.insert_(3);
		b.insert_(1);
		TSet<int> result = a.subtract(b);
		Assert::AreEqual(true, result.container == c);
	}

	TEST_METHOD(SUBTRACT_2) {
		TSet<int> a;
		TSet<int> b;
		set<int> c{ 1, 2, 3 };
		a.insert_(1);
		a.insert_(2);
		a.insert_(3);
		b.insert_(4);
		TSet<int> result = a.subtract(b);
		Assert::AreEqual(true, result.container == c);
	}

	TEST_METHOD(MULTIPLY_1) {
		TSet<int> a;
		TSet<int> b;
		set<int> c{ 1, 2, 3 };
		a.insert_(1);
		a.insert_(2);
		a.insert_(3);
		b.insert_(1);
		b.insert_(2);
		b.insert_(3);
		TSet<int> result = a.multiply(b);
		Assert::AreEqual(true, result.container == c);
	}

	TEST_METHOD(MULTIPLY_2) {
		TSet<int> a;
		TSet<int> b;
		set<int> c{ 1 };
		a.insert_(1);
		a.insert_(2);
		a.insert_(3);
		b.insert_(1);
```

```
			b.insert_(4);
			b.insert_(5);
			TSet<int> result = a.multiply(b);
			Assert::AreEqual(true, result.container == c);
		}

		TEST_METHOD(COUNTER_1) {
			TSet<int> a;
			a.insert_(1);
			a.insert_(2);
			a.insert_(3);
			Assert::AreEqual(3, a.count());
		}

		TEST_METHOD(COUNTER_2) {
			TSet<int> a;
			Assert::AreEqual(0, a.count());
		}

		TEST_METHOD(ELEMENT_1) {
			auto constr = [] {TSet<int> a; a.insert_(1); a.element(5); };
			Assert::ExpectException<std::invalid_argument>(constr);
		}

		TEST_METHOD(ELEMENT_2) {
			auto constr = [] {TSet<int> a; a.insert_(1); a.element(-5); };
			Assert::ExpectException<std::invalid_argument>(constr);
		}

		TEST_METHOD(ELEMENT_3) {
			auto constr = [] {TSet<int> a; a.insert_(1); a.element(1000); };
			Assert::ExpectException<std::invalid_argument>(constr);
		}

		TEST_METHOD(ELEMENT_4) {
			TSet<int> a;
			a.insert_(5);
			Assert::AreEqual(5, a.element(0));
		}
	};
};
```