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 Множество»

Выполнил:
Студент группы ИП-113
Шпилев Д. И.
Работу проверил:
старший преподаватель кафедры ПМиК
Агалаков А.А.

Новосибирск 2024 г.

Содержание

1. 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. Спецификация типа данных «множество»

ADT tset

Данные

Множества - это изменяемые неограниченные наборы элементов типа T. Содержимое множества изменяется следующими операциями: • Опустошить (опустошение множества); • Добавить (добавление элемента во множество); • Удалить (извлечение элемента из множества). Множество поддерживает следующую дисциплину записи и извлечения элементов: элемент может присутствовать во множестве только в одном экземпляре, при извлечении выбирается заданный элемент множества и удаляется из множества. Операции Операции могут вызываться только объектом «множество» (тип tset), указатель на который передаётся в них по умолчанию. При описании операций этот объект в разделе «Вход» не указывается.

2.1 Код программы

TSet.h

```
#pragma once
#include <set>
#include <algorithm>

template <class T>
class TSet
{
public:
	TSet() {};
	TSet(const std::set<T>& st) : m_set(st) {};
	void clear() { m_set.clear(); }
	void add(const T& item) { m_set.insert(item); }
	void remove(const T& removable) { m_set.erase(removable); }
	bool empty() const noexcept { return m_set.empty(); }
	bool belongs(const T& item) { return m_set.count(item); }
	size_t size() { return m_set.size(); }
	TSet<T> unions(const TSet<T>& st);
	TSet<T> subtract(const TSet<T>& st);
	TSet<T> multiply(const TSet<T>& st);
```

```
	T at(size_t index);

private:
	std::set<T> m_set;
};

template<class T>
inline TSet<T> TSet<T>::unions(const TSet<T>& st)
{
	std::set<T> res;
	std::set_union(m_set.cbegin(), m_set.cend(),
		st.m_set.cbegin(), st.m_set.cend(),
		std::inserter(res, res.begin()));
	return TSet<T>(res);
}

template<class T>
inline TSet<T> TSet<T>::subtract(const TSet<T>& st)
{
	std::set<T> res;
	std::set_difference(m_set.cbegin(), m_set.cend(),
		st.m_set.cbegin(), st.m_set.cend(),
		std::inserter(res, res.begin()));
	return TSet<T>(res);
}

template<class T>
inline TSet<T> TSet<T>::multiply(const TSet<T>& st)
{
	std::set<T> res;
	std::set_intersection(m_set.cbegin(), m_set.cend(),
		st.m_set.cbegin(), st.m_set.cend(),
		std::inserter(res, res.begin()));
	return TSet<T>(res);
}

template<class T>
inline T TSet<T>::at(size_t index)
{
	if (index >= m_set.size()) {
		throw std::out_of_range("Index " + std::to_string(index) + " out of
range");
	}
	auto it = m_set.begin();
	std::advance(it, index);
	return *it;
}
```

2.2 Код тестов

```
TEST_CLASS(TSetTests)
{
public:

	TEST_METHOD(TestAdd)
	{
		TSet<int> set;
		set.add(1);
		Assert::IsTrue(set.belongs(1));
		Assert::AreEqual(static_cast<size_t>(1), set.size());
	}
```

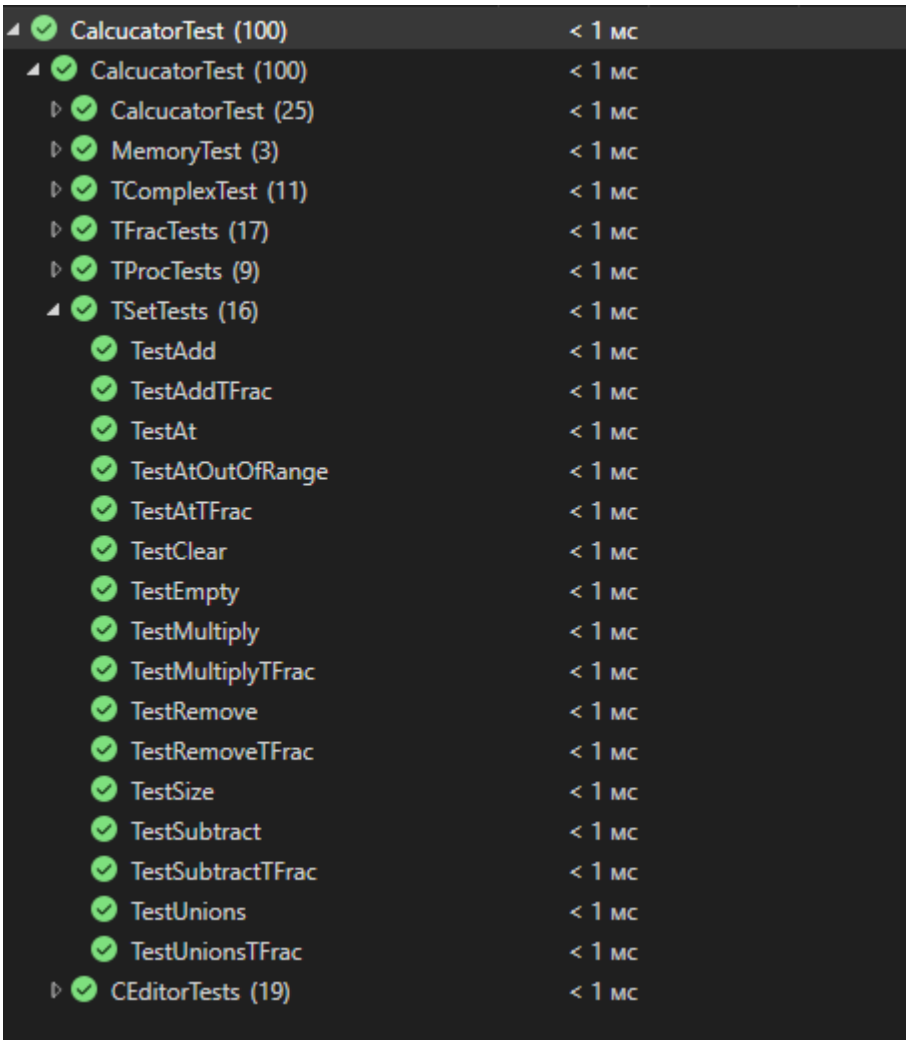
```
TEST_METHOD(TestRemove)
{
	TSet<int> set;
	set.add(1);
	set.remove(1);
	Assert::IsFalse(set.belongs(1));
	Assert::AreEqual(static_cast<size_t>(0), set.size());
}

TEST_METHOD(TestEmpty)
{
	TSet<int> set;
	Assert::IsTrue(set.empty());
	set.add(1);
	Assert::IsFalse(set.empty());
}

TEST_METHOD(TestSize)
{
	TSet<int> set;
	set.add(1);
	set.add(2);
	Assert::AreEqual(static_cast<size_t>(2), set.size());
}

TEST_METHOD(TestUnions)
{
	TSet<int> set1;
	set1.add(1);
	set1.add(2);

	TSet<int> set2;
	set2.add(2);
	set2.add(3);

	TSet<int> result = set1.unions(set2);
	Assert::AreEqual(static_cast<size_t>(3), result.size());
	Assert::IsTrue(result.belongs(1));
	Assert::IsTrue(result.belongs(2));
	Assert::IsTrue(result.belongs(3));
}

TEST_METHOD(TestSubtract)
{
	TSet<int> set1;
	set1.add(1);
	set1.add(2);

	TSet<int> set2;
	set2.add(2);
	set2.add(3);

	TSet<int> result = set1.subtract(set2);
	Assert::AreEqual(static_cast<size_t>(1), result.size());
	Assert::IsTrue(result.belongs(1));
	Assert::IsFalse(result.belongs(2));
	Assert::IsFalse(result.belongs(3));
}

TEST_METHOD(TestMultiply)
{
	TSet<int> set1;
	set1.add(1);
	set1.add(2);
```

```
        TSet<int> set2;
        set2.add(2);
        set2.add(3);

        TSet<int> result = set1.multiply(set2);
        Assert::AreEqual(static_cast<size_t>(1), result.size());
        Assert::IsTrue(result.belongs(2));
}

TEST_METHOD(TestAt)
{
        TSet<int> set;
        set.add(1);
        set.add(2);

        Assert::AreEqual(1, set.at(0));
        Assert::AreEqual(2, set.at(1));

        Assert::ExpectException<std::out_of_range>([&set] { set.at(2); });
}

TEST_METHOD(TestClear)
{
        TSet<int> set;
        set.add(1);
        set.add(2);

        set.clear();
        Assert::IsTrue(set.empty());
}

TEST_METHOD(TestAddTFrac)
{
        TSet<TFrac> fracSet;
        TFrac frac1(1, 2);
        TFrac frac2(3, 4);

        fracSet.add(frac1);
        fracSet.add(frac2);

        Assert::AreEqual((size_t)2, fracSet.size());
        Assert::IsTrue(fracSet.belongs(frac1));
        Assert::IsTrue(fracSet.belongs(frac2));
}

TEST_METHOD(TestRemoveTFrac)
{
        TSet<TFrac> fracSet;
        TFrac frac1(1, 2);
        TFrac frac2(3, 4);
        fracSet.add(frac1);
        fracSet.add(frac2);

        fracSet.remove(frac1);

        Assert::AreEqual((size_t)1, fracSet.size());
        Assert::IsFalse(fracSet.belongs(frac1));
        Assert::IsTrue(fracSet.belongs(frac2));
}

TEST_METHOD(TestUnionsTFrac)
{
        TSet<TFrac> fracSet1;
        TSet<TFrac> fracSet2;
```

```
        TFrac frac1(1, 2);
        TFrac frac2(3, 4);
        TFrac frac3(5, 6);
        fracSet1.add(frac1);
        fracSet1.add(frac2);
        fracSet2.add(frac2);
        fracSet2.add(frac3);

        TSet<TFrac> unionSet = fracSet1.unions(fracSet2);

        Assert::AreEqual((size_t)3, unionSet.size());
        Assert::IsTrue(unionSet.belongs(frac1));
        Assert::IsTrue(unionSet.belongs(frac2));
        Assert::IsTrue(unionSet.belongs(frac3));
    }

    TEST_METHOD(TestSubtractTFrac)
    {
        TSet<TFrac> fracSet1;
        TSet<TFrac> fracSet2;
        TFrac frac1(1, 2);
        TFrac frac2(3, 4);
        TFrac frac3(5, 6);
        fracSet1.add(frac1);
        fracSet1.add(frac2);
        fracSet2.add(frac2);
        fracSet2.add(frac3);

        TSet<TFrac> subtractSet = fracSet1.subtract(fracSet2);

        Assert::AreEqual((size_t)1, subtractSet.size());
        Assert::IsTrue(subtractSet.belongs(frac1));
        Assert::IsFalse(subtractSet.belongs(frac2));
    }

    TEST_METHOD(TestMultiplyTFrac)
    {
        TSet<TFrac> fracSet1;
        TSet<TFrac> fracSet2;
        TFrac frac1(1, 2);
        TFrac frac2(3, 4);
        TFrac frac3(5, 6);
        fracSet1.add(frac1);
        fracSet1.add(frac2);
        fracSet2.add(frac2);
        fracSet2.add(frac3);

        TSet<TFrac> multiplySet = fracSet1.multiply(fracSet2);

        Assert::AreEqual((size_t)1, multiplySet.size());
        Assert::IsTrue(multiplySet.belongs(frac2));
    }

    TEST_METHOD(TestAtTFrac)
    {
        TSet<TFrac> fracSet;
        TFrac frac1(1, 2);
        TFrac frac2(3, 4);
        fracSet.add(frac1);
        fracSet.add(frac2);

        Assert::IsTrue(frac1 == fracSet.at(0));
        Assert::IsTrue(frac2 == fracSet.at(1));
    }
```

```
        TEST_METHOD(TestAtOutOfRange)
        {
                TSet<TFrac> fracSet;
                TFrac frac1(1, 2);
                fracSet.add(frac1);

                Assert::ExpectException<std::out_of_range>([&fracSet]() {
fracSet.at(2); });
        }
    };
```

3. Результаты модульных тестов



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.