Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет

телекоммуникаций и информатики»

Кафедра ПМиК

Современные технологии программирования 1

Лабораторная работа №12

Абстрактный тип данных Множество

Выполнил: студент 4 курса

Ф. ИВТ, группа: ИП-711

Мартасов И. О.

Проверил: доцент кафедры ПМиК

Зайцев Михаил Георгиевич

Новосибирск, 2020

**Содержание**

1. Задание.
2. Текст программы.
3. Результаты тестирования.

**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.

**Текст программы**

TSet.cpp

#include "TSet.h"

template<class T>

TSet<T>::TSet() {}

template<class T>

TSet<T>::TSet(const TSet& q)

{

tset = q.tset;

}

template<class T>

void TSet<T>::clear(void)

{

tset.clear();

}

template<class T>

void TSet<T>::add(const T& d)

{

tset.insert(d);

}

template<class T>

void TSet<T>::remove(const T& d)

{

if (tset.count(d))

tset.erase(d);

}

template<class T>

bool TSet<T>::empty(void)

{

return tset.empty();

}

template<class T>

bool TSet<T>::contains(const T& d)

{

return tset.count(d);

}

template<class T>

TSet<T> TSet<T>::concat(const TSet<T>& q)

{

TSet<T> result;

set\_union(tset.begin(), tset.end(), q.tset.begin(), q.tset.end(), inserter(result.tset, result.tset.begin()));

return result;

}

template<class T>

TSet<T> TSet<T>::subtract(const TSet<T>& q)

{

TSet<T> result;

set\_difference(tset.begin(), tset.end(), q.tset.begin(), q.tset.end(), inserter(result.tset, result.tset.begin()));

return result;

}

template<class T>

TSet<T> TSet<T>::multiply(const TSet<T>& q)

{

TSet<T> result;

set\_intersection(tset.begin(), tset.end(), q.tset.begin(), q.tset.end(), inserter(result.tset, result.tset.begin()));

for (const T& elem : q.tset)

if (tset.count(elem))

result.add(elem);

return result;

}

template<class T>

int TSet<T>::count(void)

{

return tset.size();

}

template<class T>

T TSet<T>::element(const int& j)

{

if (j < 0 || j >= tset.size())

return T();

T it = \*next(tset.begin(), j);

return it;

}

template<class T>

TSet<T>::~TSet() {

tset.clear();

}

TSet.h

#pragma once

#include <set>

using namespace std;

template <class T>

class TSet {

set<T> tset;

public:

TSet();

TSet(const TSet& q);

void clear(void);

void add(const T& d);

void remove(const T& d);

bool empty(void);

bool contains(const T& d);

TSet<T> concat(const TSet<T>& q);

TSet<T> subtract(const TSet<T>& q);

TSet<T> multiply(const TSet<T>& q);

int count(void);

T element(const int& j);

~TSet();

};

TSet1.cpp

#include "TSet1.h"

template<class T>

TSet1<T>::TSet1() : set<T>() {}

template<class T>

void TSet1<T>::add(const T& d)

{

set<T>::insert(d);

}

template<class T>

void TSet1<T>::remove(const T& d)

{

if (set<T>::count(d))

set<T>::erase(d);

}

template<class T>

bool TSet1<T>::contains(const T& d)

{

return set<T>::count(d);

}

template<class T>

TSet1<T> TSet1<T>::concat(const TSet1<T>& q)

{

TSet1<T> result;

set\_union(set<T>::begin(), set<T>::end(), q.begin(), q.end(), inserter(result, result.begin()));

return result;

}

template<class T>

TSet1<T> TSet1<T>::subtract(const TSet1<T>& q)

{

TSet1<T> result;

set\_difference(set<T>::begin(), set<T>::end(), q.begin(), q.end(), inserter(result, result.begin()));

return result;

}

template<class T>

TSet1<T> TSet1<T>::multiply(const TSet1<T>& q)

{

TSet1<T> result;

set\_intersection(set<T>::begin(), set<T>::end(), q.begin(), q.end(), inserter(result, result.begin()));

return result;

}

template<class T>

int TSet1<T>::count(void)

{

return set<T>::size();

}

template<class T>

T TSet1<T>::element(const int& j)

{

if (j < 0 || j >= set<T>::size())

return T();

return \*next(set<T>::begin(), j);

}

template<class T>

TSet1<T>::~TSet1()

{

set<T>::clear();

}

TSet1.h

#include <set>

using namespace std;

template <class T>

class TSet1 : public set<T>

{

public:

public:

TSet1();

void add(const T& d);

void remove(const T& d);

bool contains(const T& d);

TSet1<T> concat(const TSet1<T>& q);

TSet1<T> subtract(const TSet1<T>& q);

TSet1<T> multiply(const TSet1<T>& q);

int count(void);

T element(const int& j);

~TSet1();

};

UnitTests.cpp

#include "pch.h"

#include "CppUnitTest.h"

#include <algorithm>

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

#include "../TSet/TSet1.h"

#include "../TSet/TSet.cpp"

#include "../TSet/TSet1.cpp"

#include "../../TFrac/TFrac/TFrac.h"

#include "../../TFrac/TFrac/TFrac.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1

{

TEST\_CLASS(UnitTest1)

{

public:

TEST\_METHOD(TestMethod1)

{

TSet<int> t0, t1, res;

string str = "", strRes;

for (int i = 1; i < 10; i++) {

t0.add(i);

t1.add(2 + i);

}

for (int i = 0; i < t0.count(); i++)

str += to\_string(t0.element(i)) + " ";

strRes = "1 2 3 4 5 6 7 8 9 ";

Assert::AreEqual(strRes, str);

str.clear();

for (int i = 0; i < t1.count(); i++)

str += to\_string(t1.element(i)) + " ";

strRes = "3 4 5 6 7 8 9 10 11 ";

Assert::AreEqual(strRes, str);

res = t0.concat(t1);

str.clear();

for (int i = 0; i < res.count(); i++)

str += to\_string(res.element(i)) + " ";

strRes = "1 2 3 4 5 6 7 8 9 10 11 ";

Assert::AreEqual(strRes, str);

res = t0.subtract(t1);

str.clear();

for (int i = 0; i < res.count(); i++)

str += to\_string(res.element(i)) + " ";

strRes = "1 2 ";

Assert::AreEqual(strRes, str);

res = t0.multiply(t1);

str.clear();

for (int i = 0; i < res.count(); i++)

str += to\_string(res.element(i)) + " ";

strRes = "3 4 5 6 7 8 9 ";

Assert::AreEqual(strRes, str);

res.remove(4);

str.clear();

for (int i = 0; i < res.count(); i++)

str += to\_string(res.element(i)) + " ";

strRes = "3 5 6 7 8 9 ";

Assert::AreEqual(strRes, str);

res.clear();

bool check = res.empty();

Assert::AreEqual(true, check);

}

TEST\_METHOD(TestMethod2)

{

TSet1<int> t0, t1, res;

string str = "", strRes;

for (int i = 1; i < 10; i++) {

t0.add(i);

t1.add(2 + i);

}

for (int i = 0; i < t0.count(); i++)

str += to\_string(t0.element(i)) + " ";

strRes = "1 2 3 4 5 6 7 8 9 ";

Assert::AreEqual(strRes, str);

str.clear();

for (int i = 0; i < t1.count(); i++)

str += to\_string(t1.element(i)) + " ";

strRes = "3 4 5 6 7 8 9 10 11 ";

Assert::AreEqual(strRes, str);

res = t0.concat(t1);

str.clear();

for (int i = 0; i < res.count(); i++)

str += to\_string(res.element(i)) + " ";

strRes = "1 2 3 4 5 6 7 8 9 10 11 ";

Assert::AreEqual(strRes, str);

res = t0.subtract(t1);

str.clear();

for (int i = 0; i < res.count(); i++)

str += to\_string(res.element(i)) + " ";

strRes = "1 2 ";

Assert::AreEqual(strRes, str);

res = t0.multiply(t1);

str.clear();

for (int i = 0; i < res.count(); i++)

str += to\_string(res.element(i)) + " ";

strRes = "3 4 5 6 7 8 9 ";

Assert::AreEqual(strRes, str);

res.remove(4);

str.clear();

for (int i = 0; i < res.count(); i++)

str += to\_string(res.element(i)) + " ";

strRes = "3 5 6 7 8 9 ";

Assert::AreEqual(strRes, str);

res.clear();

bool check = res.empty();

Assert::AreEqual(true, check);

}

TEST\_METHOD(TestMethod3)

{

TSet<TFrac> t0, t1, res;

string str = "", strRes;

for (int i = 0; i < 10; i++) {

t0.add(TFrac(i, i \* i));

t1.add(TFrac(i, i + 1));

}

for (int i = 0; i < t0.count(); i++)

str += t0.element(i).getFractionString() + " ";

strRes = "0/1 1/9 1/8 1/7 1/6 1/5 1/4 1/3 1/2 1/1 ";

Assert::AreEqual(strRes, str);

str.clear();

for (int i = 0; i < t1.count(); i++)

str += t1.element(i).getFractionString() + " ";

strRes = "0/1 1/2 2/3 3/4 4/5 5/6 6/7 7/8 8/9 9/10 ";

Assert::AreEqual(strRes, str);

res = t0.concat(t1);

str.clear();

for (int i = 0; i < res.count(); i++)

str += res.element(i).getFractionString() + " ";

strRes = "0/1 1/9 1/8 1/7 1/6 1/5 1/4 1/3 1/2 2/3 3/4 4/5 5/6 6/7 7/8 8/9 9/10 1/1 ";

Assert::AreEqual(strRes, str);

res = t0.subtract(t1);

str.clear();

for (int i = 0; i < res.count(); i++)

str += res.element(i).getFractionString() + " ";

strRes = "1/9 1/8 1/7 1/6 1/5 1/4 1/3 1/1 ";

Assert::AreEqual(strRes, str);

res = t0.multiply(t1);

str.clear();

for (int i = 0; i < res.count(); i++)

str += res.element(i).getFractionString() + " ";

strRes = "0/1 1/2 ";

Assert::AreEqual(strRes, str);

res.remove(TFrac(1, 2));

str.clear();

for (int i = 0; i < res.count(); i++)

str += res.element(i).getFractionString() + " ";

strRes = "0/1 ";

Assert::AreEqual(strRes, str);

res.clear();

bool check = res.empty();

Assert::AreEqual(true, check);

}

TEST\_METHOD(TestMethod4)

{

TSet1<TFrac> t0, t1, res;

string str = "", strRes;

for (int i = 0; i < 10; i++) {

t0.add(TFrac(i, i \* i));

t1.add(TFrac(i, i + 1));

}

for (int i = 0; i < t0.count(); i++)

str += t0.element(i).getFractionString() + " ";

strRes = "0/1 1/9 1/8 1/7 1/6 1/5 1/4 1/3 1/2 1/1 ";

Assert::AreEqual(strRes, str);

str.clear();

for (int i = 0; i < t1.count(); i++)

str += t1.element(i).getFractionString() + " ";

strRes = "0/1 1/2 2/3 3/4 4/5 5/6 6/7 7/8 8/9 9/10 ";

Assert::AreEqual(strRes, str);

res = t0.concat(t1);

str.clear();

for (int i = 0; i < res.count(); i++)

str += res.element(i).getFractionString() + " ";

strRes = "0/1 1/9 1/8 1/7 1/6 1/5 1/4 1/3 1/2 2/3 3/4 4/5 5/6 6/7 7/8 8/9 9/10 1/1 ";

Assert::AreEqual(strRes, str);

res = t0.subtract(t1);

str.clear();

for (int i = 0; i < res.count(); i++)

str += res.element(i).getFractionString() + " ";

strRes = "1/9 1/8 1/7 1/6 1/5 1/4 1/3 1/1 ";

Assert::AreEqual(strRes, str);

res = t0.multiply(t1);

str.clear();

for (int i = 0; i < res.count(); i++)

str += res.element(i).getFractionString() + " ";

strRes = "0/1 1/2 ";

Assert::AreEqual(strRes, str);

res.remove(TFrac(1, 2));

str.clear();

for (int i = 0; i < res.count(); i++)

str += res.element(i).getFractionString() + " ";

strRes = "0/1 ";

Assert::AreEqual(strRes, str);

res.clear();

bool check = res.empty();

Assert::AreEqual(true, check);

}

};

**Результаты тестирования**

