Министерство науки и высшего образования

Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего образования

«Новосибирский государственный технический университет»



Кафедра теоретической и прикладной информатики

Расчётно-графическое задание  
по дисциплине «Метрология, качество и тестирование программного обеспечения»

**Метрические характеристики программ и модульное тестирование**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Факультет: | ПМИ |  |  |
| Группа: | ПМИМ-01 |  |  |
| Студент: | Ничипиенко В.А. |  |  |
| Вариант: | 12 |  |  |
| Преподаватель: | Зайцев М.Г. |  |  |

Новосибирск

2022

1. **Цель работы**

Сформировать практические навыки модульного тестирования средствами Visual Studio, прогнозирования длины программы и трудоемкости ее реализации на основе спецификации.

1. **Задание**
2. Реализовать абстрактный тип данных «Множество» в соответствии со спецификацией приведённой ниже и протестировать его, используя средства модульного тестирования Visual Studio. Тестовые наборы необходимо построить на основе критериев тестирования C0,C1,C2 в зависимости от варианта задания.
3. На основе заданной спецификации программы определить метрические характеристики ПС:

* длину программы N;
* квалификационное время программирования Т;

1. По тексту программы оцените значение уровня языка реализации:

* λ;
* количество первичных ошибок .

1. Сопоставить время, потраченное на программирование, с расчетным временем программирования.

|  |  |  |  |
| --- | --- | --- | --- |
| №Варианта | Тип | Критерий тестирования | Реализация |
| 12 | char | C1,C2 | Агрегирование |

1. **Текст программы**

*TSet.h*

#pragma once

#include <set>

#include <algorithm>

#include <iostream>

using namespace std;

class TSet

{

public:

TSet(){ }

void Clear();

void Add(char item);

void Remove(char item);

bool Empty();

bool Belongs(char item);

TSet Union(TSet tset);

TSet Difference(TSet tset);

TSet Intersection(TSet tset);

int Size();

char Elem(int index);

private:

set <char> set;

};

*TSet.cpp*

#include "TSet.h"

void TSet::Clear()

{

set.clear();

}

void TSet::Add(char item)

{

set.insert(item);

}

void TSet::Remove(char item)

{

set.erase(item);

}

bool TSet::Empty()

{

if (set.empty())

{

return true;

}

else

{

return false;

}

}

bool TSet::Belongs(char item)

{

if (set.find(item) != set.end())

{

return true;

}

else

{

return false;

}

}

TSet TSet::Union(TSet tset)

{

std::set <char> newSet;

for (int i = 0; i < tset.Size(); i++)

{

newSet.insert(tset.Elem(i));

}

std::set<char> unionSet;

set\_union(set.begin(), set.end(), newSet.begin(), newSet.end(),

std::inserter(unionSet, unionSet.begin()));

TSet newTSet;

for (auto i : unionSet)

{

newTSet.Add(i);

}

return newTSet;

}

TSet TSet::Difference(TSet tset)

{

std::set <char> newSet;

for (int i = 0; i < tset.Size(); i++)

{

newSet.insert(tset.Elem(i));

}

std::set<char> differenceSet;

set\_difference(set.begin(), set.end(), newSet.begin(), newSet.end(),

std::inserter(differenceSet, differenceSet.begin()));

TSet newTSet;

for (auto i : differenceSet)

{

newTSet.Add(i);

}

return newTSet;

}

TSet TSet::Intersection(TSet tset)

{

std::set <char> newSet;

for (int i = 0; i < tset.Size(); i++)

{

newSet.insert(tset.Elem(i));

}

std::set<char> intersectionSet;

set\_intersection(set.begin(), set.end(), newSet.begin(), newSet.end(),

std::inserter(intersectionSet, intersectionSet.begin()));

TSet newTSet;

for (auto i : intersectionSet)

{

newTSet.Add(i);

}

return newTSet;

}

int TSet::Size()

{

return set.size();

}

char TSet::Elem(int index)

{

std::set<char>::iterator it = set.begin();

advance(it, index);

char elem = \*it;

return elem;

}

*UnitTestTSet.cpp*

#include "CppUnitTest.h"

#include "C:\Users\Виталий\source\repos\PSet\PSet\TSet.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTestTSet

{

TEST\_CLASS(UnitTestTSet)

{

public:

TEST\_METHOD(TestMethodClear)

{

TSet tset;

tset.Add(0);

Assert::IsFalse(tset.Empty());

tset.Clear();

Assert::IsTrue(tset.Empty());

}

TEST\_METHOD(TestMethodAdd)

{

TSet tset;

char elem = 0;

tset.Add(elem);

Assert::IsTrue(tset.Belongs(elem));

}

TEST\_METHOD(TestMethodRemove)

{

TSet tset;

char elem = 0;

tset.Add(elem);

tset.Remove(elem);

Assert::IsFalse(tset.Belongs(elem));

}

TEST\_METHOD(TestMethodEmptyTrue)

{

TSet tset;

Assert::IsTrue(tset.Empty());

}

TEST\_METHOD(TestMethodEmptyFalse)

{

TSet tset;

char elem = 0;

tset.Add(elem);

Assert::IsFalse(tset.Empty());

}

TEST\_METHOD(TestMethodBelongsTrue)

{

TSet tset;

char elem = 0;

tset.Add(elem);

Assert::IsTrue(tset.Belongs(elem));

}

TEST\_METHOD(TestMethodBelongsFalse)

{

TSet tset;

char elem = 0;

Assert::IsFalse(tset.Belongs(elem));

}

TEST\_METHOD(TestMethodUnion1)

{

TSet tset1;

TSet tset2;

TSet tset3;

tset1 = tset1.Union(tset2);

Assert::AreEqual(tset1.Empty(), tset3.Empty());

}

TEST\_METHOD(TestMethodUnion2)

{

TSet tset1;

TSet tset2;

TSet tset3;

tset1.Add(0);

tset3.Add(0);

tset1 = tset1.Union(tset2);

Assert::AreEqual(tset1.Size(), tset3.Size());

for (int i = 0; i < tset1.Size(); i++)

{

Assert::AreEqual(tset1.Elem(i), tset3.Elem(i));

}

}

TEST\_METHOD(TestMethodUnion3)

{

TSet tset1;

TSet tset2;

TSet tset3;

tset1.Add(1);

tset2.Add(0);

tset3.Add(1);

tset3.Add(0);

tset1 = tset1.Union(tset2);

Assert::AreEqual(tset1.Size(), tset3.Size());

for (int i = 0; i < tset1.Size(); i++)

{

Assert::AreEqual(tset1.Elem(i), tset3.Elem(i));

}

}

TEST\_METHOD(TestMethodUnion4)

{

TSet tset1;

TSet tset2;

TSet tset3;

tset1.Add(1);

tset1.Add(0);

tset2.Add(1);

tset2.Add(0);

tset3.Add(1);

tset3.Add(0);

tset1 = tset1.Union(tset2);

Assert::AreEqual(tset1.Size(), tset3.Size());

for (int i = 0; i < tset1.Size(); i++)

{

Assert::AreEqual(tset1.Elem(i), tset3.Elem(i));

}

}

TEST\_METHOD(TestMethodUnion5)

{

TSet tset1;

TSet tset2;

TSet tset3;

tset1.Add(1);

tset1.Add(2);

tset1.Add(3);

tset2.Add(3);

tset2.Add(4);

tset2.Add(5);

tset3.Add(1);

tset3.Add(2);

tset3.Add(3);

tset3.Add(4);

tset3.Add(5);

tset1 = tset1.Union(tset2);

Assert::AreEqual(tset1.Size(), tset3.Size());

for (int i = 0; i < tset1.Size(); i++)

{

Assert::AreEqual(tset1.Elem(i), tset3.Elem(i));

}

}

TEST\_METHOD(TestMethodDifference)

{

TSet tset1;

TSet tset2;

TSet tset3;

tset1.Add(1);

tset1.Add(2);

tset1.Add(3);

tset2.Add(3);

tset2.Add(4);

tset2.Add(5);

tset3.Add(1);

tset3.Add(2);

tset1 = tset1.Difference(tset2);

Assert::AreEqual(tset1.Size(), tset3.Size());

for (int i = 0; i < tset1.Size(); i++)

{

Assert::AreEqual(tset1.Elem(i), tset3.Elem(i));

}

}

TEST\_METHOD(TestMethodIntersection)

{

TSet tset1;

TSet tset2;

TSet tset3;

tset1.Add(1);

tset1.Add(2);

tset1.Add(3);

tset2.Add(3);

tset2.Add(4);

tset2.Add(5);

tset3.Add(3);

tset1 = tset1.Intersection(tset2);

Assert::AreEqual(tset1.Size(), tset3.Size());

for (int i = 0; i < tset1.Size(); i++)

{

Assert::AreEqual(tset1.Elem(i), tset3.Elem(i));

}

}

TEST\_METHOD(TestMethodSize)

{

TSet tset;

tset.Add(1);

tset.Add(2);

tset.Add(3);

Assert::AreEqual(tset.Size(), 3);

}

TEST\_METHOD(TestMethodElem)

{

TSet tset;

tset.Add(1);

tset.Add(2);

tset.Add(3);

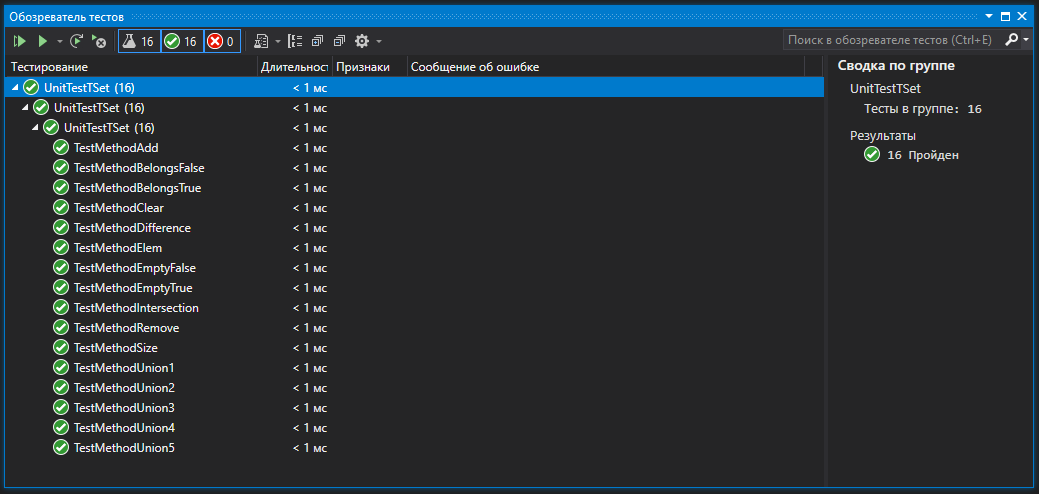
Assert::AreEqual(tset.Elem(0), (char)1);

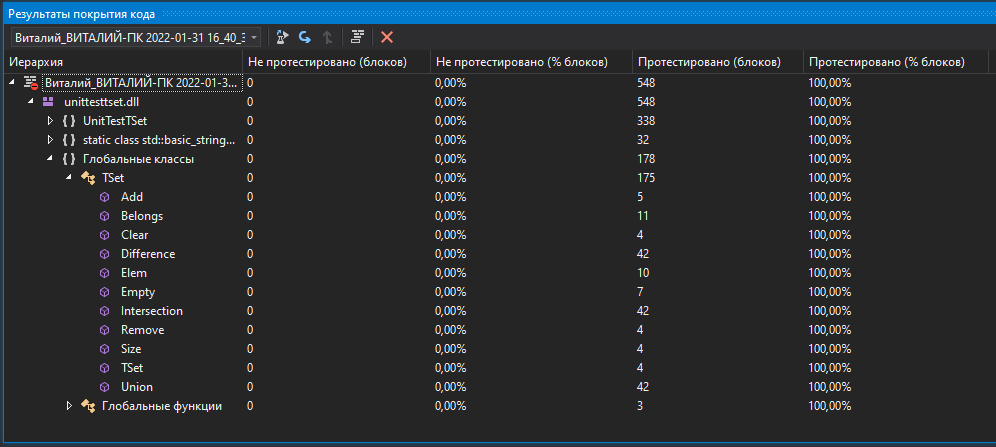
}

};

}

1. **Результаты модульного тестирования**





1. **Метрические характеристики программы**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n\*2 | n1 | n2 | n | N1 | N2 | N |
| 8 | 31 | 37 | 68 | 48 | 46 | 94 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| V\* | V | length | T | λ | B0 |
| 33.2193 | 572.222 | 261.103 | 547.603 | A 1.92849 | 0.190741 |

1. **Вывод по выполненной работе**

В ходе выполнения работы были сформированы практические навыки модульного тестирования средствами Visual Studio, прогнозирования длины программы и трудоемкости ее реализации на основе спецификации.